Zeus TrafficScript Overview and Reference

Version 7.1

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

Sales: +44 (0)1223 568555
Main: +44 (0)1223 525000
Fax: +44 (0)1223 525100
Email: info@zeus.com
Web: www.zeus.com

Zeus Technology, Inc. (U.S.)
1875 South Grant Street – Suite 720
San Mateo
CA 94402
United States of America

Phone: 1-888-ZEUS-INC
Fax: (866) 628-7884
Email: info@zeus.com
Web: www.zeus.com

Copyright Notice

Table of Contents

Introduction

1.1 Introducing Zeus Traffic Manager

Zeus Traffic Manager is a high-availability, application-centric traffic management and load balancing product. It provides control, intelligence, security and resilience for all your application traffic. Zeus Traffic Manager is intended for organizations hosting valuable business-critical services, such as TCP and UDP-based services like HTTP (web) and media delivery, and XML-based services such as Web Services.

Zeus Traffic Manager's unique architecture ensures it can handle large volumes of network traffic efficiently. Its TrafficCluster™ scalability allows you to add more front-end traffic managers or back-end servers to your cluster as the need arises. The cluster size is unlimited, and the performance of the traffic manager grows in line with the performance of the hardware.

Zeus Traffic Manager is a highly capable solution which can also be adapted and extended as new requirements arise. Using the TrafficScript language you can write tailored traffic management rules to inspect, manage and route requests and responses. TrafficScript rules can manage connections in any TCP or UDP-based protocol.

Zeus Traffic Manager is secure out-of-the-box, and is hardened against intrusion and Denial-of-Service (DoS) attacks. It incorporates the fastest and strongest SSL encryption technologies, and can efficiently decrypt and encrypt large numbers of SSL connections. TrafficScript rules, security policies and other content-based calculations can be applied to the encrypted request while retaining full end-to-end security.

For critical, high-availability solutions, Zeus Traffic Manager offers TrafficCluster™ redundancy. This allows you to have unlimited numbers of active and standby front-end servers. If one of your active machines fails, a standby traffic manager will be automatically brought into action; in the case of subsequent failure, more standby servers are available to take up the load. This ensures that there is no single point of failure in the system.

A centralized web-based administration console monitors and manages each traffic management unit in your service infrastructure.

1.2 The TrafficScript language

Zeus Traffic Manager can be customized using custom traffic management rules. These rules are created using a scripting language called TrafficScript.

TrafficScript rules are executed whenever a new connection or network request is received, and whenever it receives a response from a node. The rules can inspect the incoming and outgoing data in the connection, and other aspects such as the remote client address.

The TrafficScript rules can then modify the request or response (for example, rewriting the URL or headers in an HTTP request), set session persistence parameters, or decide how to route the request to the most appropriate pool.

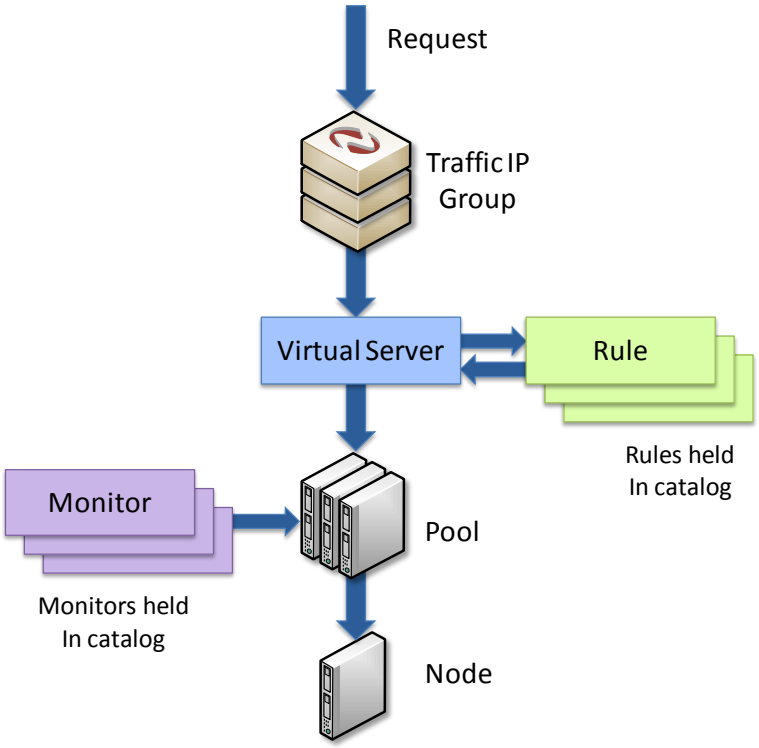

Fig. 1. Analyzing and managing traffic using TrafficScript rules

This makes it possible to control precisely how traffic is using rules designed to meet to your own hosting requirements.

Using TrafficScript it is extremely easy to perform any of these traffic management tasks:

- Inspect incoming or outgoing traffic and rewrite it fully or in part as desired.
- Restrict a website to a certain range of IP addresses.
- Apply selective management to elements such as web spiders.
- Enable or disable functions for a given request or response (such as compression)
- Retry request that generate errors a maximum number of times.
- Future-proof your services against any change in the back-end components of the system.
- Work around broken links and content on your website.

1.2.1 TrafficScript Examples

The following TrafficScript rule can be used with HTTP requests. It handles the request as follows:

- Requests for 'www.zeus.co.uk' are rewritten to 'www.zeus.com';
- Requests for .jsp pages are routed to a set of application servers (the pool named `JSPServers`);
- Requests for URLs beginning '/secure' are only allowed during office hours.

```
# Rewrite host header if necessary
if( http.getHeader( "Host" ) == "www.zeus.co.uk" ) {
  http.setHeader( "Host", "www.zeus.com" );
}

$path = http.getPath();

# Give .jsp requests to the 'JSPServers' pool
if( string.endsWith( $path, ".jsp" ) ) {
  pool.use( "JSPServers" );
}

# Deny access to /secure outside office hours
if( string.startsWith( $path, "/secure" ) ) {
  if( sys.time.hour() < 9 || sys.time.hour() >= 18 ){
    connection.discard();
  }
}
```

The next rule can be used with HTTP responses. It processes the response as follows:

- If the status code is 404 or 5xx, retry the request a maximum of 3 times;
- If the response contains references to `www.zeus.co.uk`, rewrite it by changing these references to `www.zeus.com`.

```
$code = http.getResponseCode();
if( $code == 404 || $code >= 500 ) {
  if( request.getRetries() < 3 ) {
    # Avoid the current node when we retry,
    # if possible
    request.avoidNode( connection.getNode() );
    request.retry();
```

```
  } else {
    http.sendResponse( "312 Redirect", "text/plain",
      "", "Location: /" );
  }
}

# We're only going to process text/html responses, so
# break out of the rule if the response is of a
# different type...
if( http.getResponseHeader(
      "Content-Type" ) != "text/html" ) break;

# Note: need to prevent node giving compressed
# responses Should use
# http.removeHeader( "Accept-Encoding" ) in
# request rule
$response = http.getResponseBody();

if( string.contains( $response,"www.zeus.co.uk" ) ) {
  $response = string.regexsub( $response,
         "www.zeus.co.uk", "www.zeus.com", "g" );
  http.setResponseBody( $reponse );
}
```

1.3 Application of Rules

There are many occasions when you might use a TrafficScript rule.

Rules are used by a virtual server to choose a pool to handle a request. The rule can inspect any part of the request, possibly modify it, and decide which pool should handle the request.

- You can use a rule to dictate session persistence information to a pool. After inspecting the request the rule can use the `connection.setPersistenceKey()` function to provide a string to persist on. This string is used by the **Universal** session persistence method to identify the session the request belongs to.
- Rules can be used to check the response from the server and modify it, or even retry the request (if possible) if a transient error was detected.
- If you use various back-end systems with different presentation styles or even different protocols, rules can be used to integrate them into a single, coherent and consistent service. Incoming requests can be rewritten into the format suitable for the required service, and responses can be rewritten into a single, consistent form.
- For example, HTTP requests that involve a database lookup can be rewritten into SOAP request for a Web Service; the XML response can then be transformed into a suitable HTML document to return via HTTP.
- Rules can override the classes assigned to a connection by the virtual server or the pool. This way, they can specify custom behaviour for each connection; connections to a slow resource can be given a longer response time tolerance for example. Classes you can assign in this way include *Service Level Monitoring*, *Session Persistence* and *Bandwidth Management*.
- *Service protection classes* can use rules. If you have associated a service protection class with your virtual server, it inspects the incoming packets. The class may use a rule to check the packet for a match with known web worms or viruses. This rule is executed before the main processing of the virtual server is carried out.

1.4 Using a TrafficScript Rule

TrafficScript rules are stored in the Rules Catalog. You can create rules here, modify or duplicate them, and delete unused rules as required.

A virtual server processes traffic, and you can configure the virtual server to execute one or more rules from the catalog each time it receives a new connection.

This way, several different virtual servers can use the same rule, and modifications to the rule take effect on all virtual servers.

To use the TrafficScript functionality you need to:

1. Create a new rule in the catalog.
2. Configure your virtual server to use the rule (**Virtual Servers**, edit the server, then go to **Rules**).

1.4.1 Create a Rule in the Catalog

1. Click the **Configure** button in the tool bar. Go to the **Catalogs** tab, then click **Rules Catalog**.
2. Below **Create New Rule**, enter a name for your rule. To write the rule in TrafficScript select the **TrafficScript Language** option, then click **Create Rule**.

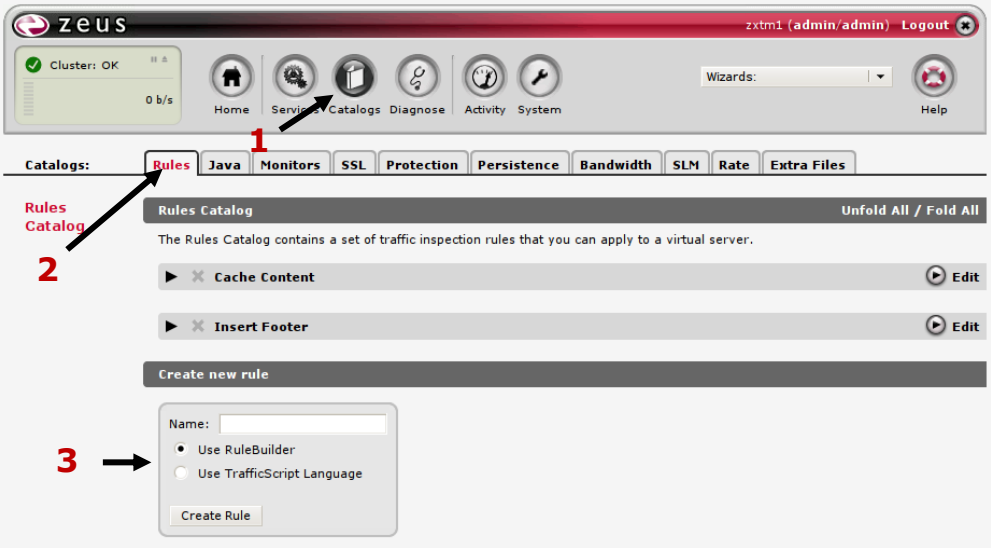

Fig. 2. 3 simple steps to create a new rule

3. This takes you to the TrafficScript editing page. Here you can enter a text description of the rule, and type the rule. You may wish to write the rule in a separate text editor before pasting it in.

   You can click the **TrafficScript Help** link to view the quick function reference, and use the **Check Syntax** button to check your rule.

4. When you have finished, click the **Update** button to save your edits.

Note that some TrafficScript functions are only appropriate in request rules or in response rules. For example, a function that modifies a parameter of a request will have no effect if used in a response rule (as the request has already been submitted to a node). Please refer to the documentation for each function in this manual, or in the online help.

1.4.2 Configure a Virtual Server to Use a Rule

The new rule has been successfully created, but is not yet being used by any virtual servers. To configure a virtual server to use the rule, follow the instructions below.

1. Go to the **Virtual Servers > Edit** page for your virtual server. You can do this by clicking the **Configure** button, then the **Virtual Servers** tab. Click on the name of your virtual server.

2. Now click on **Rules**. This presents you with a list of request rules and a list of response rules currently in use for that virtual server. You can choose new rules to add to each list from the drop-down selection at the end of each list.

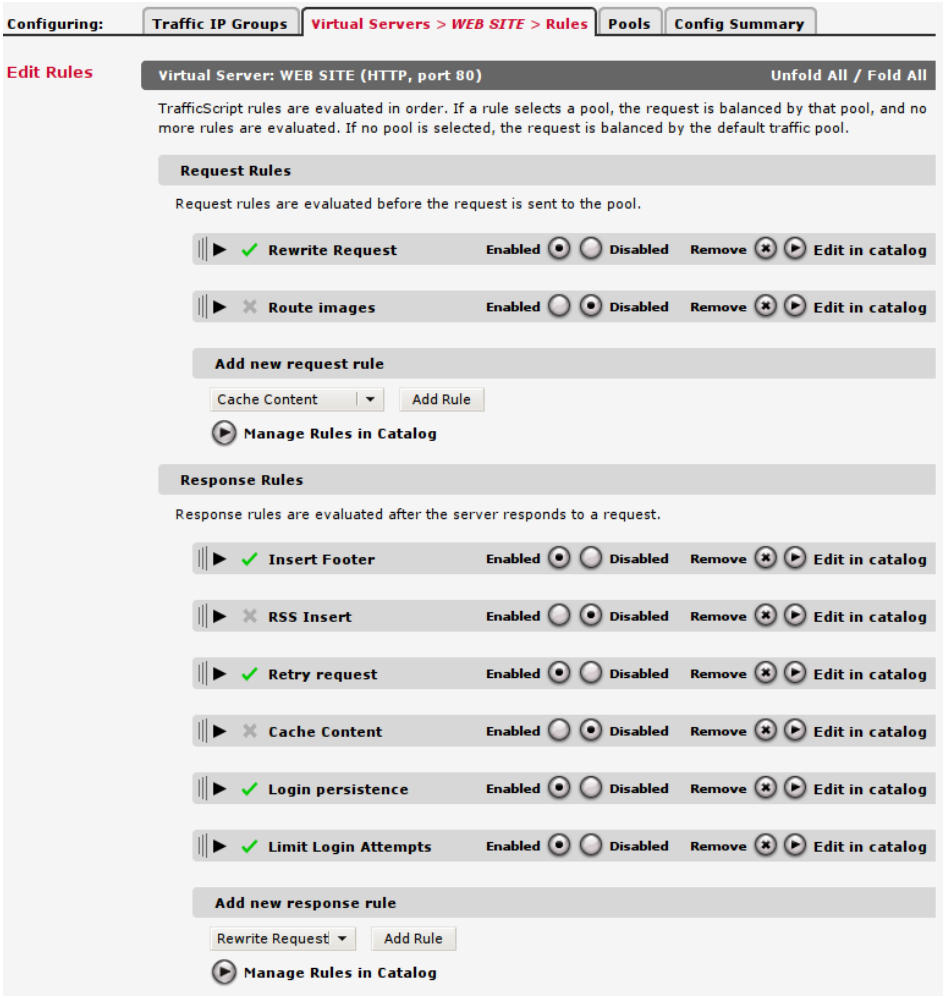

Fig. 3. Applying a rule to a virtual server

3. Rules are executed in a specified order. If the first rule does not make a final decision about a request or response, the second rule is tested, etc. You can drag rules up and down to re-order the list.

4. For non-http protocols, you can specify whether the rule should be executed just once (against the first request or response), or against every request and response in the protocol dialogue.

This option is not necessary for HTTP virtual servers because HTTP is a single request-response protocol, and requests within a keepalive connection are processed independently.

You can test the effect of a new rule by enabling and disabling it for your test virtual server.

TrafficScript Syntax

TrafficScript is the scripting language provided by Zeus Traffic Manager. An administrator can create TrafficScript rules to process requests, implementing suitable logic to ensure that requests are handled in the most appropriate way.

TrafficScript is similar to many other programming or scripting languages, such as C or Perl. This chapter describes the syntax of the language.

2.1 Statements

A command in TrafficScript is called a ***statement***. Each statement ends with a semicolon ';', and a TrafficScript rule typically contains several statements.

```
http.setHeader( "Host", "secure.mysite.com" );
pool.use( "mypool" );
```

Any text between a '#' and the end of the line is called a ***comment*** and is ignored.

```
# Write a message to the error log file
log.info( "Starting to run rule now!" );

$body = http.getBody( ); # get the POST data
```

2.2 Constants

TrafficScript allows you to specify integer, floating point and string values:

- **Integers**: sequences of digits, such as '23'; hex format (0xff) and octal format (\377) are also supported.
- **Floating point**: decimal point (3.14) and scientific (5.6e-2) notations are both supported.
- **Strings**: strings are character sequences, enclosed by " (double) quotes or ' (single) quotes.

  In double-quoted strings, special characters can be escaped using the standard '\' notation (for example, "\n" is a newline character), using octal notation ("\012" is also a newline) or using hexadecimal notation ("\x0A" is a newline).

  In single-quoted strings, no escaping is performed. For example, '\n' is not converted to the newline character; in fact, it is not possible to embed a newline character in a single quoted string.

In both types of string, for improved readability, strings can be broken across lies using a single '\' followed by a newline:

```
# The following two lines both create the string 'Hello world'

$single = 'Hello \
world';

$double = "Hello \
world";
```

You need to be aware of escaping rules when writing regular expressions in TrafficScript. For example, in a double-quoted TrafficScript string, the character '\' automatically escapes the next character, so if you want a literal '\' in your string, you need to double-escape it '\\'. For this reason, regular expressions are often written using single-quoted strings.

2.3 Variables

Variables are used in TrafficScript to store values while the rule is executed.

A ***variable*** can store an integer, floating point or string value. The value is interpreted as the correct type depending on its context.

Variables have global scope, and exist for the duration of the execution of the rule. Values stored in variables are discarded when the rule completes.

Variable names always start with the '$' character, and the value of a variable is set by the assignment operator '='.

Variables are immutable in TrafficScript i.e they are never directly modified by functions, a copy of the argument is modified.

```
$path = http.GetPath();
$bytes = connection.getDataLen();
$pi = 3.14157;
```

2.4 Expressions

Expressions in TrafficScript are created from combinations of literal values, variables, evaluated functions and operators. Expressions are evaluated when the rule is executed.

Expressions can be used to:

- Construct complex strings from several different values;
- Create complex tests for 'if' conditions or 'while' loops;
- Perform mathematical calculations.

An expression can be used anywhere a literal value (or variable) would be suitable.

You'd normally see expressions:

- In assignment statements, assigning a value to a variable;
- In conditions – 'if/else' statements and 'while' loops;
- As function arguments.

For example:

```
$message = "The URL path is " . http.GetPath();
$four = 2 + 2;

# Sets $ratio to "75%" (for example)
$ratio = ( $a / ($a + $b) * 100 ) . "%";

$contentLength = http.getHeader( "Content-Length" );
if( $contentLength > 1024 * 1024 ) {
  log.warn( "Large request body: ".$contentLength );
}
```

2.4.1 Operators

Expressions are constructed from operands (variables, literal values, etc.) and operators. Operators perform calculations or tests on their operands.

Operands in an expression are automatically promoted to the appropriate type: string, integer or float in accordance with the typecasting rules described in section 2.4.2. The following operators can be used:

Mathematical

The operators '+', '-', '*' and '/' treat their operands as integers or doubles and add, subtract, multiply or divide them.

The prefix operator '-' promotes its operand to an integer or double and returns its negation.

```
4 + 7.5 * $a
```

```
-$b / $c - 1
```

The modulus operator '%' takes integer operands, and calculates the remainder after division.

```
7 % 3    # Returns 1
3 % 7    # Returns 3
```

String Concatenation

The operator '.' promotes its operands to strings and concatenates them.

```
"The message is " . $bytes . " bytes long."
( $a / ($a + $b ) * 100 ) . " percent"
```

The '.=' operator appends its second operand to the first:

```
$message = "The name is ";
$message .= "Bond, James Bond";
```

Comparison

The operators '==', '!=', '>', '<', '>=' and '<=' compare their operands and return 0 (false) or 1 (true). If both arguments are strings, a string comparison is performed; otherwise the operands are promoted to integers or floats and compared. The typecasting promotion rules are described in section 2.4.2

```
1 > 3.14      # false
"99" > 100    # false (performs integer comparison)
"99" > "100"  # true (performs string comparison)
$a == $b      # are the values of $a and $b the same?
```

Boolean

The operators '&&' and '||' perform boolean 'and' and 'or' tests. The prefix operator '!' performs a Boolean 'not' test.

These operators treat their operands as either 'true' or 'false', and return 1 (true) or 0 (false):

- A non-zero number operand or non-empty string operand is '*true*';

- A zero number operand or empty string operand is '*false*'.

```
"foo" && !0                 # true
( 1 < 2 ) && ( 3 < 4 )      # true
$a || $b                    # true if $a or $b is true
```

Increment/Decrement

$foo++	increment $foo after it has been referenced
$foo--	decrement $foo after it has been referenced
++$foo	increment $foo before it has been referenced
--$foo	decrement $foo before it has been referenced
$foo += 5	add 5 to the value of $foo ie $foo = $foo + 5
$foo -= 5	subtract 5 from the value of $foo ie $foo = $foo - 5

Bitwise operators

The operators '&' (bitwise-AND), '|' (bitwise-OR) and '^' (bitwise-XOR) perform bitwise operations on their integer arguments. Strings and floats are typecast to integers using the typecasting rules in section 2.4.2

```
0x1234 & 255                # 0x34
1 | 2 | 4                   # 7
1 ^ 3                       # 2
```

The prefix operator '~' (bitwise-NOT) performs a bitwise NOT on its integer argument.

```
~1 & 0xffff                 # 65534
```

The bitwise left- and right-shift operators ('<<' and '>>') perform left and right bit-shifts on their integer argument.

```
1 << 2                  # 4
2 >> 1                  # 1
```

Precedence

Complex expressions follow the standard rules of precedence. Parentheses '(' and ')' can be used to group sub-expressions.

2.4.2 Type Casts in TrafficScript

Variables in TrafficScript can contain data of various types: integer, floating point (double) or string. TrafficScript automatically casts (converts) values and variables into the correct type when you evaluate an expression or call a function:

Value:	Cast to *integer*:	Cast to *double*:	Cast to *string*:
14 (integer)	14	14.0	"14"
3.25 (double)	3	3.25	"3.25"
3.75 (double)	4	3.75	"3.75"
"abcde"(string)	0	0.0	"abcde"
"3.25" (string)	3	3.25	"3.25"
"3.75" (string)	4	3.75	"3.75"
"14str" (string)	14	14.0	"14str"
"3.2.7" (string)	3	3.2	"3.2.7"

Strings and doubles are rounded up or down to the nearest integer value when they are cast to integers.

```
$int = 10;
$double = 2.71828;

string.len( $int );    # casts to string, returns 2
string.len( $double ); # casts to string, returns 7

# Set $string to "10, 2.71828"
$string = $int . ", " . $double;
```

```
# Convert $string to a number, and add 4:
$r = $string + 4;  # $r is 14
```

2.5 Conditionals

TrafficScript provides 'if' and 'if/else' statements for conditional execution. The condition is an expression, which is evaluated.

The return value of the expression determines whether the condition is 'true' or 'false':

- A non-zero number or non-empty string is '*true*';
- A zero number or empty string is '*false*'.

```
if( <condition> ) {
  <statement list>
}
```

or

```
if( <condition> ) {
  <statement list>
} else {
  <statement list>
}
```

The condition is evaluated. If it is *true* (the value is non-zero, or is a non-empty string), the first statement list is executed. If it is *false*, the second statement list is used (in the case of an 'if/else' conditional).

```
$path = http.getPath();
if( string.startsWith( $path, "/secure" ) ) {
  pool.use( "secure pool" );
} else {
  pool.use( "non-secure pool" );
}
```

2.6 Loops

A body of code can be executed a number of times using a loop. TrafficScript supports 'for', 'do' and 'while' loops.

2.6.1 'for' loops

A 'for' loop contains an initialization step, a test and an increment step, bracketing the body of code to be executed on each iteration:

```
for( <initialization> ; <condition> ; <increment> ) {
   <statement list>
}
```

For example:

```
for( $count = 0; $count < 10; $count++ ) {
   log.info( "In loop, count = " . $count );
}
```

This loop will print the message 10 times, with `$count` running from 0 to 9.

2.6.2 'while' loops

The 'while' loop evaluates a condition, and while the condition is 'true', it executes the enclosed block of code:

```
while( <condition> ) {
   <statement list>
}
```

For example,

```
$count = 0;
while( $count < 10 ) {
   log.info( "In loop, count = " . $count );
   $count = $count + 1;
}
```

This loop will print the message 10 times, with `$count` running from 0 to 9.

2.6.3 'do' loops

The 'do' loop executes the enclosed block of code, then checks the condition. It repeats the code while the condition evaluates to true:

```
do {
   <statement list>
while( <condition> );
```

For example:

```
$count = 0;
do {
   log.info( "In loop, count = " . $count );
   $count = $count + 1;
} while( $count < 10 );
```

This loop will print the message 10 times, with `$count` running from 0 to 9.

TrafficScript contains a number of helper functions to manipulate complex data, such as HTTP requests or long strings. Consequently, it is rarely necessary to use loops in TrafficScript.

2.7 Other flow control

The 'break' and 'continue' statements can be used to restart or exit loop or rules processing.

Inside a while loop, 'break' causes execution of the loop to stop; execution continues to the statement after the loop. 'continue' causes the loop code to restart.

Inside a rule (i.e., outside any containing loops), 'break' causes execution of the rule to stop; execution proceeds to the next TrafficScript rule. 'continue' causes the rule to be restarted from the beginning.

For example:

```
# We're only interested in processing HTTP text/html
# responses...

$mime = http.getResponseHeader( "Content-Type" );
if( !string.startsWith( $mime, "text/html" )) break;

# proceed with processing for text/html...
```

2.8 Complex Data Types

2.8.1 Arrays

An array in TrafficScript is a structured variable that stores a list of values. You can define a list of names for example using the following code:

```
$array = [ "Alex", "Matt", "Oliver", "Laurence" ];
```

The values in this array can then be looked up:

```
$someone = $array[0];
log.info($someone);
```

This will cause the string "Alex" to be printed to the event log. TrafficScript has functions that make it easy to work with array structures. If you wanted to print all of the names stored in `$array` using a **for loop**, you need to know the number of elements that it stores. The `array.length()` function will return the number of elements in an array:

```
$array = [ "Alex", "Matt", "Oliver", "Laurence" ];

$arraylen = array.length($array);
log.info("My array has " . $arraylen . " elements.\n");

for ( $i = 0; $i < $arraylen; $i++ ){
   log.info ( "Element #" . $i . " " . $array[$i]);
}
```

When applied as a rule, the code above will cause the following output in the event log:

There is a slightly easier way to do this – using a **foreach loop**. Starting with the first element, it initializes the variable specified on the left side of the in operator with a new element as it goes through each element in the array. The code below will produce output similar to the above:

```
$array = [ "Alex", "Matt", "Oliver", "Laurence" ];
$i = 0;

log.info ("My array has " . array.length($array) . " elements.\n");

foreach ( $element in $array ) {
   log.info( "Element #" . $i . " " . $element );
   $i++;
}
```

For more information on array-specific functions, please refer to the TrafficScript Guide in the Zeus Traffic Manager Admin UI.

2.8.2 Hashes

A hash in TrafficScript is similar to an array, but instead of storing a list of values it stores a list of key/value pairs. Hashes are sometimes referred to as associative arrays. You can define a hash using the following code:

```
$hash = [ "orange" => "fruit",
          "apple" => "fruit",
          "cabbage" => "vegetable",
          "pear" => "fruit" ];
```

In order to print all of the keys and values stored in the hash, you can first get a list of keys in the form of an array – using the `hash.keys()` function - that you can then use in a **foreach loop** to print all of the values:

```
foreach ( $key in hash.keys($hash)){
   log.info("Key: " . $key . "; Value: " . $hash[$key] . ";");
}
```

Combining the above two samples of code will result in the following output in the event log:

2.8.3 The global associative array

This is the first of two persistent associative arrays that TrafficScript can access.

The global array can be accessed using the `data.set()` and `data.get()` functions. Data that is set in this array is persistent, and can be read from a later script. This array is of fixed size (the size is defined by the global setting '**trafficscript!data_size**'); when it fills up, you cannot add further entries without first removing some.

Maintaining the array

Elements are added and looked up using the `data.set()` and `data.get()` functions. Individual elements can be deleted using the `data.remove()` function.

You can determine the amount of memory in use by the global array using `data.getMemoryUsage()`, and delete all entries in the array using the `data.reset()` function. You can delete a subset of the entries using `data.reset( "prefix" )`.

For example, if you wish to store several different types of data globally, you should use a consistent prefix to start the name of each key. Then, if you need to free memory, you can easily delete all of the data that is stored one particular purpose (for example, a global cache that grows continually, but can safely be deleted and reconstructed if necessary).

Example: An indexed array

You can use the global associative array to create an indexed array named 'myarray' as follows:

```
# Declare a subroutine to calculate factorials
sub factorial( $n ) {
   if( $n == 0 ) return 1;
   return $n*factorial( $n-1 );
}

# Put entries into the array
$c = 0;
while( $c <= 10 ) {
   $msg = "Did you know that ". $c ."! is ". factorial( $c ) ."?" ;
   data.set( "myarray".$c, $msg );
   $c++;
}

# Look up several entries.  Note: the 1000th entry is empty
$msg = "";
$msg .= "Index 5:    ".data.get( "myarray5" )."\n";
$msg .= "Index 10:   ".data.get( "myarray10" )."\n";
$msg .= "Index 1000: ".data.get( "myarray1000" )."\n";

# delete the entire array (but no other data stored by data.set)
data.reset( "myarray" );

http.sendResponse( "200 OK", "text/plain", $msg, "" );
```

2.8.4 The connection-local array

This is the second of two associative arrays that TrafficScript can access.

When processing a connection, it is sometimes useful to store information calculated in one rule for retrieval in a later rule. You can do this using a connection-local associative array, using the `connection.data.set()` and `connection.data.get()` functions.

Information stored in this way can only be retrieved by a TrafficScript rule that is processing the same connection, and all information is destroyed (and the memory freed) when the connection completes.

2.8.5 Libraries

TrafficScript rules that contain subroutines can be used as libraries. Take the following rule for example:

```
sub headbug(){
   # Prints each header to the event log.

   $headers = http.listHeaderNames();

   foreach ($header in $headers){
      log.info( $header . ": " . http.getheader($header));
   }
}
```

The `http.listHeaderNames()` function returns an array of header names that were sent in the HTTP request. If we save this as a rule named **foo**, we can then create another rule called **bar** which imports the `headbug` subroutine:

```
import foo;

foo.headbug();
```

Applying this rule to a virtual server will cause the names and values for each header of each request that the virtual server processes to be logged to the event log. We can modify the first rule/library so that it creates a hash of each of the headers:

```
sub headbug(){
   # Prints each header to the event log.

   $headers = http.listHeaderNames();

   foreach ($header in $headers){
      $headhash = [ $header => http.getheader($header) ];
      log.info( $header . ": " . $headhash[ $header ] );
   }
}
```

Although this library does the same thing, it does it slightly differently in that it creates a data structure that can be used later. If we want to be able to use this structure later however, we need to get it out of the subroutine. To do this, we need to get the `headbug` subroutine to return the `$headhash` data structure, then modify the **bar** rule:

```
# foo (library)

sub headbug(){
   # Prints each header to the event log.

   $headers = http.listHeaderNames();

   $headhash = [];

   foreach ($header in $headers){
      $headhash[ $header ] = http.getheader($header);
      #log.info( $header . ": " . $headhash[ $header ] );
   }

   return($headhash);
}
```

```
# (bar rule)

import foo;

$headhash = foo.headbug();

foreach ($header in hash.keys($headhash)){
   log.info( $header . ": " . $headhash[ $header ]);
}
```

2.9 Functions

A function performs an action, and returns a value. Functions are used to provide useful capabilities to TrafficScript.

TrafficScript contains a large number of functions to manipulate data or manage the current request. For ease of use, TrafficScript functions are grouped into families. For example, functions that operate on HTTP requests all begin with '`http.`', like '`http.getHeader()`' or '`http.setBody()`'.

A function is called using its name, followed by a pair of parentheses '`( )`'. Many functions take one or more values as parameters, and these are listed inside the parentheses.

```
connection.discard();
```

```
http.setHeader( "Cookie: type=chocolate" );
$hello = string.append( "Hello", " ", "world", "!" );
```

Function names are not case sensitive. So, the function 'lang.todouble()' can be invoked using 'lang.toDouble()' or 'Lang.ToDouble()' as well as 'lang.todouble()'.

Chapter 6 (Function Reference) describes the various TrafficScript functions.

2.10 Escaping Regular Expressions

Several TrafficScript functions take regular expressions as arguments. TrafficScript uses the PCRE regular expression library.

Regular expressions may contain a number of special characters:

.	matches any single character
?	indicates that the previous expression is optional
*	matches any number of the previous expression
^	matches the beginning of a string
$	matches the end of a string

If you wish to match a literal '.', or other special character in your regular expression, you escape it with a '\' character.

Note: Like many other scripting languages, double-quoted TrafficScript strings use '\' as an escape character, so to put a regular expression like "^192\.168\." into a TrafficScript double-quoted string, you must double-escape the '\' character:

```
# Sets the regex string as ^192\.168\. ; the two examples
# below have the same effect
$regex = "^192\\.168\\.";
$regex = '^192\.168\.';
if ( string.regexMatch( $ip, $regex ) ) {
    # IP is on 192.168.* network
}
```

Note that it's not often necessary to use regular expressions in TrafficScript;

- string.IPMaskMatch(), string.Contains(), string.startsWith() and string.endsWith() can be used to search strings.
- string.replace() and string.replaceAll() can be used to search-and-replace within strings.

2.11 Creating new subroutines in TrafficScript

You can create new TrafficScript subroutines to improve the efficiency of your software. Once created, these routines can be used in rules or procedures just like the predefined functions.

2.11.1 Syntax

The syntax to create a new subroutine is:

```
sub function_name ($var1, $var2)
{
      <code>
      return "return value"
}
```

Subroutines can be called just like you would call any normal function:

```
$ret = function_name( $foo, $var );
```

This would return the value given by the subroutine into the variable '$ret'.

Subroutine position and name restrictions

Name restrictions: user subroutine names cannot be the same as the TrafficScript built-in functions.

Subroutines can be declared above or below where they are used. For example, the following code will be perfectly correct:

```
test();
sub test()
{
     log.info("Attention! Test running");
}
```

Local variables

Variables in the subroutines are local to that section and won't affect the result when used out of the subroutine.

For instance, the following lines of code

```
$var = "abc";
```

```
sub function()
{
      log.info($var);
}
function();
```

…will print out an empty string, as `$var` is not available within the subroutine.

$1 to $9 variables

$1 to $9 are always treated as global variables, so they can be used to return extra data from subroutines, in addition to the (optional) return statement that returns a single value.

For example, the following code will print "Hello World":

```
sub greetings()
{
     $1="Hello ";
     return "World"
}
$ret = greetings();
log.info($1 . $ret)
```

2.12 Request and Response rules

TrafficScript rules are assigned to a virtual server. A rule can be used as a *request rule* or a *response rule*.

- Request rules are executed when the first request data is received from a client. A request rule can read further data if the client's request is not complete.

  Once the request rules complete, a connection to a back-end server is made and the request data is streamed to the server.

- Response rules are executed when the first data in the response is read from the back-end server. A response rule can then read further data from the server, and can flush the current data to the client.

  Once the response rules complete, the remaining data is streamed to the client.

2.12.1 Processing multiple requests and responses

If the client sends another request down the same network connection the traffic manager can be configured to run the request and response rules again against this new request. This is configured via the run once/run every time setting for each rule in the virtual server configuration. You can use the `request.endsAt()` and `request.endsWith()` functions to

parse persistent connections that contain multiple requests and responses. Refer to sections 3.2.1 and 4.6 for more details.

Note that the run once/run every time option is not relevant for HTTP connections. Even though a client may send several HTTP requests down the same persistent connection, they are automatically separated into single connections internally.

2.12.2 Specialized protocol handing functions

Processing HTTP

The HTTP protocol is fully understood by the traffic manager. On a new HTTP connection, request rules are not started until all the HTTP headers have been received. Specialized TrafficScript functions (such as `http.getPath()` and `http.setHeader()`) are available to parse and manipulate the request.

When an HTTP response is received from a back-end server, it does not execute the response rules until it has received all the HTTP headers in the response. Again, specialized TrafficScript functions (such as `http.setResponseHeader()`, `http.setResponseCookie()`) are available to manipulate the response.

You cannot use the lower-level connection, request and response functions such as `request.getLine()` or `response.set()` to modify an HTTP request. Use the equivalent specialized HTTP functions instead.

Although multiple HTTP requests can be submitted down a single TCP connection (using keepalives and pipelining), the traffic manager transparently separates these requests and handles each individually. For this reason, the option to run rules once or every time is not relevant to HTTP requests.

HTTPS traffic that is decrypted by the traffic manager is handled in exactly the same way. The HTTP payload requests can be inspected and manipulated just as if they were sent in plain text.

Other specialized protocols

TrafficScript offers high-level functions for several other protocols, including SIP and RTSP. Always use the high-level functions to read and write request data where possible, rather than lower level functions such as `request.get()` or `response.set()`.

2.12.3 Processing other protocols

Other protocols (TCP and UDP based) can be managed using the functions `request.get()` and `set()` and `response.get()` and `set()`, and related functions to read requests.

Note: UDP is a connectionless protocol. To set a UDP response, you should use the TrafficScript function `connection.close( $data, 0 );`.

2.13 The state machine in detail

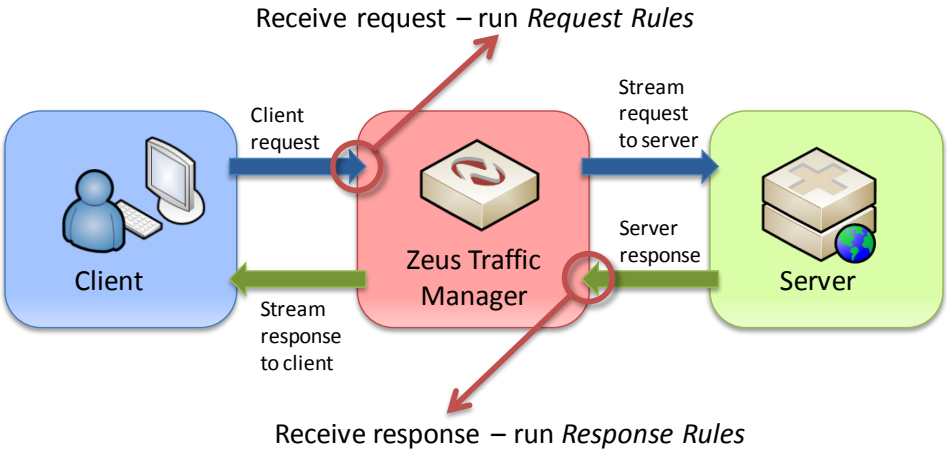

Fig. 4. Process of rules in TrafficScript

2.13.1 Controlling the state machine

The following TrafficScript functions can be used to control the state machine when processing requests and responses:

Function	Notes
`pool.use()`	When used in a request rule, aborts all rules processing and specify the pool to give the request to.
`request.sendResponse()` `http.sendResponse()`	When used in a request rule, specifies the response to send to the client. The current request is discarded and no data is sent to any back-end servers. When the request rules finish, the response rules are run on the provided response.
`response.close()`	In a response rule, close the connection to the server. When the response rules complete, pending response data is sent to the client and the traffic manager then waits for a new request from the client.

`request.endsAt()` `request.endsWith()`	In a request rule, use these functions to extract individual requests from the incoming datastream and process them synchronously in a request-response manner. For some protocols, a client may send several requests in one go. These functions can be used to process the requests one-by-one.
`request.retry()`	In a response rule, retry the request if possible. All request data that was read before and during a request rule is cached. A request can be retried if all the request data was read and cached; it can't be retried if data was subsequently streamed between the client and the server before a response was received.
`response.flush()`	In a response rule, flush the current response data. Use response.get() or response.getLine() to read further response data. This function can be used to manually stream data from the server to the client, ensuring that the response rule does not complete until the response has completed.
`connection.close()` `connection.discard()`	These functions can be used to abort a connection, optionally providing a response. The abort is immediate – no further rules are run at either the request or response stage.

Sample TrafficScript Rules

3.1 Routing by Content Type

This example inspects the URL in an HTTP request. It decides which pool to send the request to, based on the value of the URL.

Suppose your website uses a number of different technologies, including Microsoft ASP pages and Sun℠ JSP™ as well as static HTML content. Microsoft Windows® machines serve the ASP pages, Sun servers handle JSP, and a set of cheap commodity Linux servers serve the static content. You want to direct your traffic to different servers depending on the specific content.

Set up a pool for each of these groups called `windows`, `sun` and `linux` respectively. The following rule directs network traffic according to the type of content.

```
$path = http.getPath();
if( string.endsWith( $path, ".jsp" )) {
     pool.use( "sun" );
}    else if( string.endsWith( $path, ".asp" )) {
     pool.use( "windows" );
} else {
     pool.use( "linux" );
}
```

3.2 Restricting Access Based on the Time of Day

This example only allows access to a particular service during office hours (between 9am and 6pm, Monday to Friday). It discards all connections that occur outside these times.

```
$dayofweek = sys.time.weekDay();
$hourofday = sys.time.hour();

# $dayofweek: Sunday is 1, Saturday is 7
# $hourofday: office hours are between 9am and 5:59pm
if( $dayofweek == 1 ||
    $dayofweek == 7 ||
    $hourofday < 9  ||
    $hourofday >= 18 ) {
  log.warn( "Warning: access out of hours!" );
  connection.discard();
}
```

In practice, it may be more appropriate to direct restricted traffic to a separate 'error pool' of servers rather than just dropping the connection without warning. The servers in the error pool would be configured to return an appropriate error message before closing the connection. The procedure for doing this depends on the protocol being balanced.

3.3 Customer Prioritization

This example inspects the cookie in an HTTP request. It uses the value of the cookie to determine which pool to direct the request to. One pool is faster than the other because it contains machines that are reserved for premium users.

A company has a customer base divided into "gold" and "silver" membership. It wishes to give priority to the "gold" customers and has five servers, `yellow`, `green`, `blue`, `black` and `purple`.

Two pools are created: `standard`, for the "silver" customers, containing machines `yellow`, `green` and `blue`; and `premium`, for the "gold" customers, which includes all five of the servers. Thus `black` and `purple` are only available to the "gold" customers.

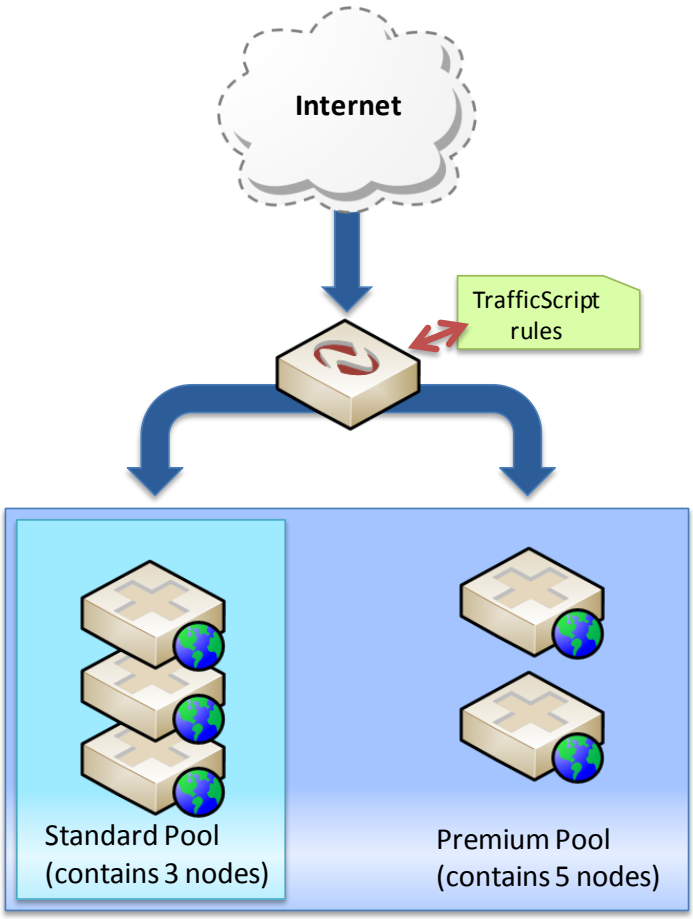

Fig. 5. Customer prioritization scheme inserting cookies using TrafficScript

The site uses a cookie login system, with the customer type encoded in the cookie. Different membership levels can be detected, and sent to the correct pool:

```
$cookie = http.getHeader( "cookie" );
if( string.contains( $cookie, "gold" )) {
   pool.use( "premium" );
} else {
   pool.use( "standard" );
}
```

3.4 Routing Based on XML Traffic

TrafficScript includes support for parsing XML documents using XPath, an industry-standard language used to query XML documents.

XML documents are used by SOAP-based protocols such as Web Services, and enable complex data to be exchanged and understood automatically without user intervention.

An XML document is organized into a tree structure of *nodes*[1]. Each node may contain a piece of data, or other nodes. XPath can navigate through these nodes to extract specific data from the XML document; this data can then be used to make routing decisions on the traffic.

The XPath 1.0 specification is available at http://www.w3.org/TR/xpath .

3.4.1 Example: Google Search Request

The Google™ search engine has a Web Services interface that accepts SOAP requests for search queries. A request for a search for Zeus Technology would consist of an HTTP POST containing the following XML body data:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <namesp1:doGoogleSearch xmlns:namesp1="urn:GoogleSearch">
         <key xsi:type="xsd:string">googleUniqueID</key>
         <q xsi:type="xsd:string">Technology</q>
         <start xsi:type="xsd:int">0</start>
         <maxResults xsi:type="xsd:int">10</maxResults>
         <filter xsi:type="xsd:boolean">false</filter>
         <restrict xsi:type="xsd:string"/>
         <safeSearch xsi:type="xsd:boolean">false</safeSearch>
```

[1] Note that these are unrelated to the traffic manager back-end nodes.

```
            <lr xsi:type="xsd:string"/>
            <ie xsi:type="xsd:string">latin1</ie>
            <oe xsi:type="xsd:string">latin1</oe>
        </namesp1:doGoogleSearch>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note that the SOAP body contains a 'doGoogleSearch' node. This contains the parameters of the search request.

An Internet service may implement or proxy doGoogleSearch requests and the traffic manager may be used to manage the traffic to this service.

For example, it may be necessary to split doGoogleSearch requests according to the specified maximum number of results. If maxResults is greater than 100, the request is to be sent to pool googleLarge; otherwise it should be sent to pool google.

A TrafficScript rule can use the functions xml.XPath.MatchNodeSet() and xml.XPath.MatchNodeCount() to query the SOAP request body and test XML nodes:

```
# Read the entire body of the SOAP/HTTP request
$body = http.getBody( 0 );

# XML parameters lie in the "urn:GoogleSearch" XML
# namespace:
$googlens = "xmlns:googlens=\"urn:GoogleSearch\"";

# Test for the presence of a "doGoogleSearch" node.
# If present, get the value of the "maxResults"
# parameter and choose the pool

if( xml.XPath.MatchNodeCount( $body, $googlens,
    "//googlens:doGoogleSearch" ) ) {

  $maxResults = xml.XPath.MatchNodeSet( $body, $googlens,
    "//googlens:doGoogleSearch/maxResults/text()" );

  if( $maxResults >= 100 ) {
    pool.use( "googleLarge" );
  } else {
    pool.use( "google" );
  }
}
```

3.5 Authenticating User Access

The following rule uses HTTP Basic authentication to ask the remote client for a username and password. It checks the client's username, password and IP address using a local web server running an authentication application.

The rule looks for a header called `Authorization`. This should contain a string containing the word `Basic`, followed by a base-64 encoded string. When decoded, this string is of the form `username:password`.

If the string is malformed or the username or password is absent, the rule returns a "401 Authorization Required" message to the client. This will cause the user's web browser to prompt them for a username and password.

If a username and password have been supplied, the rule uses these details, together with the DNS name of the client, to query the local web server using the `http.request.get()` function. It expects a "200 OK" response from the web server to indicate success. Any response code other than 200 results in the rule sending the user the response "403 Forbidden".

Note that if the user is successfully authenticated, the rule performs no positive action with the request. It will be passed on to any other rules the virtual server is using, or its default pool.

```
# Determine the username and password, and send a
# '401 Authorization Required' header if there isn't
# one.

$authheader = http.getHeader( "Authorization" );

# Decode the Authorization header; it starts with
# 'Basic', followed by a base64 encoded
# username:password string

if( string.startsWith( $authheader, "Basic " ) ) {
  $encuserpasswd = string.skip( $authheader, 6 );
  $userpasswd = string.base64decode($encuserpasswd);

  $i = string.find( $userpasswd, ":" );
  $user = string.substring( $userpasswd, 0, $i-1 );
  $password = string.skip( $userpasswd, $i+1 );
}


# If the client did not provide an Authorization
# header, indicate that authorization is required
if( $user == "" ) {
```

```
   http.sendResponse( "401 Authorization required",
      "text/html", "Please login\n",
      "WWW-Authenticate: Basic realm=\"secure
        server\"" );
}

# Check the supplied username and password by
# querying a local access server with the username,
# password and remote host.

$rhost = net.dns.resolveIP(connection.getRemoteIP());

$querystring = "user=" . string.escape( $user ) .
   "&passwd=" . string.escape( $password ) .
   "&rhost=" . string.escape( $rhost );

http.request.get(
   "http://server/auth.cgi?$querystring" );

if( $1 != 200 ) {
   # access was denied
   http.sendResponse( "403 Forbidden",
      "text/html", "Access denied\n", "" );
}

# The user is allowed access
```

This rule could be optimized to cache responses from the local web server, using the `data.set()` and `data.get()` functions.

3.6 Synchronizing requests and responses

This sample assumes that we are managing a simple line-based protocol. Each request is one line long, delimited by new-line ('\n'). Each response may be very long, but is finished by a single line containing a dot (".\n").

This protocol is similar to many simple line-based protocols like POP, IMAP or SMTP, although they each have more sophisticated ways if indicating when a response has finished. The protocol is persistent – there may be many requests and responses within a single connection.

A simple client-first or server-first virtual server can be used to manage this protocol.

Imagine that we wish to intercept a particular request and return a response directly from the traffic manager without forwarding the request to the back-end server, but we want to

keep the connection open and forward other requests to the server. In this example, we'll permit all requests other than "HACKME\n", for which "403 Go Away!\n.\n" will be returned

A first attempt might be as follows:

```
# get the request
$req = request.getLine();

if( $req == "HACKME\n" ) {
   request.sendResponse( "403 Go away!\n.\n" );
}
```

This simple filter has great potential for being subverted. A determined hacker could:

- Try sending the requests byte-by-byte to fool any parsing code (this would not work in this case);
- Try sending two requests in one packet – the first is valid, the second is 'HACKME'. Our filtering code will let the entire packet through;
- Send one request and a partial "HACK" in one packet, then send "ME\n" in a second packet when he receives the response from the first request. Our sample filter won't recognise the fragmented "HACKME" request.

To avoid these potential errors, it's necessary to synchronise individual requests, processing each request in turn and buffering up any additional data for the next request:

```
# get the request
$req = request.endsWith( "\n" );

if( $req == "HACKME\n" ) {
   request.sendResponse( "403 Go away!\n.\n" );
}
```

Using `request.endsWith()` causes data to be read up to, and including, the '\n'. Just this data will then be processed, going through the request and response stage of the state machine. When a response has been received from the server, the next will start to be processed, which may contain data that was postponed from the previous request.

This simply and effectively guards against attempts to subvert the parser by sending two requests in one, or by sending partial requests.

This solution has an unfortunate side effect. Imagine if a user sent two requests in quick succession; the first request caused the server to send a large response; the second request was the 'HACKME' which causes the traffic manager to send a response itself.

If the HACKME request is received while the server is still responding to the previous request, HACKME request will be processed and send a response directly. This response may be interleaved with the server's response, resulting in a corrupt datastream back to the client.

The solution is to synchronize responses in the datastream, so that processing of the next request is delayed until the previous request has completed. This can be achieved with a simple response rule:

```
$res = response.getLine();
while( $res != ".\n" ) {
            response.flush( string.len( $res ) );
            $res = response.getLine();
}
```

The response rule will not complete until a line containing ".\n" is received. When it completes, all remaining response data is flushed to the client and the next request will be processed.

3.7 Streaming HTTP responses

The low-level function `response.flush()` should not be used to manage HTTP responses; the `http.stream.*` functions should be used instead.

If you wish to generate or modify an HTTP response, the simplest way to do so is to read the response in its entirety (using `http.getResponseBody()`), manipulate it and then write it using `http.setResponseBody()`. However, this can be inefficient when managing large HTTP responses:

- Data is not written to the client until the response rule completes. If it takes time to read the entire response and manipulate it, this can potentially cause the client to timeout and close the connection.
- The entire response needs to be managed in memory; this can be very memory-inefficient. The memory is not discarded until the rule has completed and the data is written to the client.

As an alternative, you can stream HTTP responses, reading and writing them in smaller quantities. Response data will be written as soon as it is available, ensuring lower memory use and better performance.

The following functions manage HTTP streaming:

Function	Notes
`http.stream.startResponse()`	This function should be called before any data is written to the client. The headers for the response as assembled and written to the client. No response header manipulation can be performed after this point.
`http.stream.readResponse()`	Reads and returns a portion of the HTTP response body, limited by length or a delimiter character.
`http.stream.writeResponse()`	Writes the supplied body data to the client, but holds the connection open for additional body data if required.
`http.stream.finishResponse()`	Indicates that response streaming has finished. Rules processing is halted and any remaining response data is sent to the client.

The following TrafficScript rule processes HTML responses line by line, making a simple substitution:

```
$status = http.getResponseCode();
if( $status != 200 ) break;

$type = http.getResponseHeader("Content-Type");
if( !string.startsWith( $type, "text/") ) break;

http.stream.startResponse( $rcode, $type );

while( $line = http.stream.readResponse( 2048, "\n" ) ) {
   $line = string.replaceAll( $line, "zeus", "ZEUS" );
   http.stream.writeResponse( $line );
}

http.stream.finishResponse();
```

3.8 Managing FTP connections

This sample TrafficScript rule manages incoming FTP connections. It implements a proxy for the login stage, waiting until the remote user has provided a suitable username and password.

This rule should be configured as a 'run every time' request rule:

```
$req = string.trim( request.endswith( "\n" ) );

if( string.regexmatch( $req, "USER (.*)" ) ) {
   connection.data.set( "user", $1 );
   $msg = "331 Password required for ".$1."!!\r\n";
   request.sendresponse( $msg );
   break;
}

if( !string.regexmatch( $req, "PASS (.*)" ) ) {
   # Are we connected?
   if( connection.getNode() ) { break; }
   request.sendresponse( "530 Please log in!!\r\n" );
   break;
}

$user = connection.data.get( "user" );
$pass = $1;

# In this case, we'll permit any password that is
# the uppercase version of the username
# Do your own authentication here; for example,
# call a remote server with http.request.get

if( string.uppercase( $user ) != $pass ) {
   request.sendresponse(
      "530 Incorrect user or password!!\r\n" );
   break;
}

# now, replay the correct request against a new
# server instance, disconnecting from any FTP server
# we are already connected to
response.close();

connection.data.set( "state", "connecting" );
request.set(
           "USER anonymous\r\nPASS ".$user."\r\n" );

# select the pool we want...
if( $user == "ftp@zeus.com" ) {
   pool.select( "Zeus FTP pool" );
}
if( $user == "ftp@customer.com" ) {
```

```
   pool.select( "Customer FTP pool" );
}
# the default pool is 'discard', so other users
# are dropped
```

This rule needs to be configured as a run-every-time response rule:

```
if( connection.data.get("state") == "connecting" ) {
   # We've just connected.  Slurp the first line
   # (the serverfirst banner), the second line (the
   # 331 need password) and then replace the
   # serverfirst banner

   $first = response.getLine();
   $second = response.getLine( "\n", $1 );
   $remainder = string.skip( response.get(), $1 );
   response.set( $first.$remainder );
   connection.data.set( "state", "" );
}
```

Remember to configure a server-first banner for the FTP virtual server.

Troubleshooting

4.1 Overview

Writing and debugging complex TrafficScript rules is an involved process. This chapter lists some useful techniques.

4.2 Checking Syntax

A syntax error occurs if you make an error in your TrafficScript rule that prevents the traffic manager from fully understanding the rule.

When editing a rule in the UI, you can check the syntax at any time using the **Check Syntax** button on the **Edit Rule** page.

You can also check the syntax of a TrafficScript rule from the command line using the `zeus.zxtm` binary as follows:

```
$ export ZEUSHOME=/usr/local/zeus
$ $ZEUSHOME/zxtm/bin/zeus.zxtm --rulesyntaxcheck rulefile
Compilation failed, with 1 error
Error: line 20: Wrong number of arguments to function pool.use

        pool.use( );
                  ^
```

Note that TrafficScript is not case-sensitive. Function names and variables can be referred to using any combination of case.

4.3 Debugging Rules

The TrafficScript function '`log.info()`' can be used to log a message to the error log (*ZEUSHOME*`/log/errors`).

For example, the following code:

```
if( string.contains( http.getPath(), "root.exe" ) ) {
  log.info( "Discarding potential nimda attack" );
  connection.discard();
}
```

… will append the following message to your error log file:

```
$ export ZEUSHOME=/usr/local/zeus

$ tail $ZEUSHOME/log/errors

[20/Dec/2008:10:24:46 +0000] INFO   rules/RuleName  rulelogmsginfo
vservers/VSname  Discarding potential nimda attack
```

You can also inspect your error log file by viewing the 'Event Log' on the Zeus Admin Server.

When you are debugging a rule, you can use `log.info()` to print out progress messages as the rule executes. The `log.info()` function takes a string parameter; you can construct complex strings by appending variables and literals together using the '.' operator:

```
$msg = "Received ".connection.getDataLen()." bytes.";
log.info( $msg );
```

The functions `log.warn()` and `log.error()` are similar to `log.info()`. They prefix error log messages with a higher priority - either "`WARN`" or "`ERROR`". The Event Log viewer can be used to filter out low-priority messages.

You should be careful when printing out connection data verbatim, because the connection data may contain control characters or other non-printable characters. You can encode data using either '`string.hexEncode()`' or '`string.escape()`'.

You should use '`string.hexEncode()`' if the data is binary, and '`string.escape()`' if the data contains readable text with a small number of non-printable characters.

4.4 Request and Response rules

It is very easy to write a request or response rule which can stall a connection. If you call a function like `request.getLine()`, the connection will block until it receives another line of input. If you've mis-parsed the connection and the client or server are not going to send any more input, the connection will stall indefinitely (until it is timed out).

You must take great care that you correctly parse a protocol if you write a detailed handler for it. Your implementation of the handler will determine precisely how the protocol is handled. When developing such a handler, several techniques are useful:

- `log.info()` can be used to emit verbose debugging information, for example, when each rule starts and when it ends;

- A system call tracer like `strace` or `truss` can be used to monitor exactly what data is being read and written to the network.

  The script `zxtm/bin/trace` in your installation directory can assist with this – run `trace --help` for more information.

- `tcpdump` or `ethereal` can be used to monitor what the client and server is sending, but be aware that if a rule is blocked reading from the client, it will not notice that the server has sent any data.

- The connection dump page in the UI lists all current and some recent connections, and gives an indication of the state that they are in.

4.5 Special note about pool.use and pool.select

The `pool.use()` and `pool.select()` TrafficScript functions are used to decide which pool should be used to send the connection to a back-end server. They each take one argument, the name of the pool to use.

By default, the pool.use() and pool.select() TrafficScript functions only accept a string literal as the pool name:

```
pool.use( "my webserver pool" );
```

This allows the traffic manager to inspect a rule and determine precisely which pools are referenced by any virtual servers that use the rule. This information is used as follows:

- To 'type check' the usage of a pool, to ensure that only configuration settings relevant to the protocol the pool is managing are displayed, and to ensure that the same pool is not used by to virtual servers that manage different traffic protocols.
- To determine which pools are in use, so it can monitor the behavior of the nodes.
- To display the pools referenced by a Virtual Server, in the configuration summary, front page and other parts of the UI.
- To determine when a pool is no longer used, so that error information can be removed.

However, in some limited circumstances, it is efficient to use a variable to specify the desired pool.

```
if( $poolname == "pool1" ) {
   pool.use( "pool1" );
} else if( $poolname == "pool2" ) {
   pool.use( "pool2" );
} else if( $poolname == "pool3" ) {
```

```
   pool.use( "pool3" );
} else if( $poolname == "pool4" ) {
   pool.use( "pool4" );
} # etc.
```

It would be much more preferable to write the following:

```
pool.use( $poolname );
```

You can enable this behavior by enabling the **trafficscript!variable_pool_use** setting on the **Other Settings** section of **Global Settings** page in the **System** section of the Admin Server.

Please be aware that if you do so, the traffic manager will not be able to determine accurately which pools are referenced by a rule, and this will limit the internal consistency checks and error monitoring that it performs for those pools.

Function Reference

5.1 TrafficScript Core Functions

TrafficScript core functions provide the basic function set for the TrafficScript language, including support for the core TrafficScript types, mathematical functions, and date and time manipulation.

The core functions are grouped into several families:

- **array.**: These functions facilitate the use and manipulation of arrays within TrafficScript.
- **hash.**: These functions facilitate the use and manipulation of hashes (or key/value pairs) within TrafficScript.
- **json.**: These functions provide the ability to convert between TrafficScript arrays/hash variables and JavaScript Object Notation (JSON) strings.
- **lang.**: These functions deal with language-specific tasks like forced type conversions;
- **math.**: These functions provide mathematical operations;
- **string.**: These functions operate on strings and other sequences of data.
- **sys.**: These functions return operating-system related parameters, such as the hostname.
- **sys.time.**: These functions are used to format time values.

Recall that function names are not case sensitive. So, the function '`lang.todouble()`' can be invoked using '`lang.toDouble()`' or '`Lang.ToDouble()`' as well as '`lang.todouble()`'.

Many functions take parameters and return values with specific types. TrafficScript casts variables and values to the appropriate type as described in section 2.4.2 (Type Casts in TrafficScript).

5.1.1 array.append(array1, array2)

Appends each element of array2 to the end of array1. Note that this behaviour is different to that of array.push(), which would add one new element to the end of array1 containing array2. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# Append 4, 5 and 6 to the array
$numbers = [ 1, 2, 3 ];
array.append( $numbers, [ 4, 5, 6 ] );
# $numbers is now [ 1, 2, 3, 4, 5, 6 ]
```

See also: array.push

5.1.2 array.contains(array, value)

Returns whether or not the supplied array contains the specified value as an element. Note that it will not match elements inside sub-arrays.

Sample Usage

```
$array = [ "one", "two", [ "three", "four" ] ];
# true
array.contains( $array, "one" );
# false
array.contains( $array, "three" );
```

See also: array.filter

5.1.3 array.copy(array)

Returns a copy of the supplied array. This is semantically equivalent to `$arraycopy = $array`, however it will warn if the variable passed to it is not an array.

Sample Usage

```
# Loop around a sorted version of an array without
# permanently sorting it
$arr = [ "London", "Berlin", "Lisbon", "Paris" ];
foreach( $val in array.sort( array.copy( $arr ) )) {
   log.info( $val );
}
# $arr will be unsorted here
```

See also: array.create

5.1.4 array.create(size, [default])

Creates a new array of the specified size and optionally fills the array with the default data.

Sample Usage

```
# Create an array of length 20 with all the elements
# set to zero.
$zeroarray = array.create( 20, 0 );
```

See also: array.resize, array.copy

5.1.5 array.filter(array, pattern, [flags])

Removes elements from the supplied array that do not match the pattern. The optional 'flags' parameter contains a string of single-letter options. The following options are supported:

- 'i', meaning 'case insensitive' - letters in the pattern match both upper and lower case letters.
- '!', meaning negate - elements of the array that match the regular expression are removed.

The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# Get the extension headers for this request
$headers = http.listHeaderNames();
$extensions = array.filter( $headers, "^X-", "i" );
```

5.1.6 array.join(array, [separator])

Concatenates all the elements of the supplied array into a string separated by the separator. The elements will be separated by a space if no separator is supplied. Note that a warning will be printed should the supplied array itself contain any arrays or hashes.

Sample Usage

```
# Print a comma-separated list of names
$names = [ "alice", "bob", "mallory" ];
log.info( array.join( $names, ", " ) );
```

See also: string.split

5.1.7 array.length(array)

Returns the length of the supplied array

Sample Usage

```
# See how many HTTP headers there are
$headers = http.listHeaderNames();
log.info( "There are "
          . array.length( $headers )
          . " headers in this request" );
```

5.1.8 array.pop(array)

Removes the last element of the supplied array and returns it as the result of the function.

Sample Usage

```
$stack = [];
array.push( $stack, 3 );
array.push( $stack, 2 );
array.push( $stack, 1 );
# $stack is now [ 3, 2, 1 ];
# Prints 1 2 3
while( array.length( $stack ) > 0 ) {
   log.info( array.pop( $stack ) );
}
```

See also: array.unshift, array.push, array.shift

5.1.9 array.push(array, value)

Adds the supplied value to the end of the supplied array. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# Append 4 to the array
$numbers = [ 1, 2, 3 ];
array.push( $numbers, 4 );
```

See also: array.shift, array.unshift, array.pop

5.1.10 array.resize(array, size, [default])

Resizes the supplied array to the specified size. If the size of the array is being increased and the default parameter is specified then the new elements added to the array will be set to the default parameter. If the new size is smaller than the original size then the appropriate number of elements will be removed from the end of the array. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# We're only interested in the first
# 10 lines of body data - but fill in the
# rest with blank lines if there are fewer
# than 10 lines.
$body = array.resize( http.getBodyLines(), 10, "" );
```

See also: array.create

5.1.11 array.reverse(array)

Reverses the elements of the supplied array. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
$array = [ 1, 2, 3, [ 4, 5 ] ];
# array will be [ [ 4, 5 ], 3, 2, 1 ]
array.reverse( $array );
```

See also: array.sort

5.1.12 array.shift(array)

Removes the first element of the supplied array and returns it as the result of the function.

Sample Usage

```
$array = [ 1, 2, 3, 4 ];
# Empty an array from the front while printing
# its contents
while( array.length( $array ) > 0 ) {
   log.info( array.shift( $array ) );
}
```

See also: array.unshift, array.push, array.pop

5.1.13 array.sort(array, [reverse])

Sorts the supplied array alphanumerically. If the reverse parameter is supplied then the array will be sorted in reverse. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# Sort the keys of a hash before iterating over them
foreach( $key in array.sort( hash.keys( $hash ) ) ) {
   log.info( $key . " maps to " . $hash[$keys] );
}
# Sort these numbers alphanumerically
$sorted = array.sort( [ 1, 2, 3, 4, 10, 11 ] );
# The result will be [ 1, 10, 11, 2, 3, 4 ]
```

See also: array.sortNumerical

5.1.14 array.sortNumerical(array, [reverse])

Sort the supplied array numerically in ascending order. If the reverse parameter is supplied then the array will be sorted in descending order. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# Sort an array of numbers
$numbers = [ 2, 10, "3", "4", 24, "11" ];
array.sortNumerical( $numbers );
# $numbers is now [ 2, "3", "4", 10, "11", 24 ]
```

See also: array.sort

5.1.15 array.splice(array, offset, length, [values])

Replace elements in the supplied array. Any elements between offset and length will be removed from the array and any extra values specified after the length parameter will be inserted into the array at that location. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
$numbers = [ 0, 1, 2, 3, 4, 5 ];
# Starting from element 2, remove one element
# and insert "a" and "b" into the array
array.splice( $numbers, 2, 1, "a", "b" );
# $numbers is now [ 0, 1, "a", "b", 3, 4, 5 ]
# Starting at element 0, remove 3 elements
array.splice( $numbers, 0, 3 );
# $numbers is now [ "b", 3, 4, 5 ]
```

See also: array.filter

5.1.16 array.unshift(array, value)

Adds the supplied value to the front of the supplied array. The return value of this function can be used as the argument to another function to perform multiple operations on the same array.

Sample Usage

```
# Prepend 1 to the array
$numbers = [ 2, 3, 4 ];
array.unshift( $numbers, 1 );
```

See also: array.shift, array.push, array.pop

5.1.17 hash.contains(hash, key)

Returns whether the supplied hash contains a particular key.

Sample Usage

```
# See if bob is in the hash of users
# and if so, whether his password matches
if( hash.contains( $users, "bob" ) ) {
   if( $users["bob"] == $password ) {
      # Access granted
   } else {
      # Access denied
   }
} else {
    # User does not exist
}
```

See also: hash.keys

5.1.18 hash.count(hash)

Returns the number of items in the hash.

Sample Usage

```
# See how many HTTP headers there are
$headers = http.getHeaders();
log.info( "There are "
          . hash.count( $headers )
          . " headers in this request" );
```

5.1.19 hash.delete(hash)

Deletes the specified key from the hash. The return value of this function can be used as the argument to another function to perform multiple operations on the same hash.

Sample Usage

```
$hash = [ "orange" => "fruit",
          "apple" => "fruit",
          "cabbage" => "vegetable",
          "pear" => "fruit" ];
hash.delete( $hash, "cabbage" );
# keys will be orange, apple and pear
$keys = hash.keys( $hash );
```

See also: hash.keys

5.1.20 hash.empty(hash)

Removes all the values from the hash. The return value of this function can be used as the argument to another function to perform multiple operations on the same hash.

Sample Usage

```
# Empty the contents of a hypothetical set object
sub set.empty( $set ) {
   # You cannot do '$set = [];' here because that
   # would be reassigning the input variable
   hash.empty( $set );
}
set.empty( $my_set );
```

See also: hash.keys

5.1.21 hash.keys(hash)

Returns an array containing the keys that map to values in the supplied hash data structure.

Sample Usage

```
# Get the set of IPs that have connected to the site
$ips = data.get( "connected_ips" );
$ips[request.getRemoteIP()] = 1;
data.set( "connected_ips", $ips );
$connected_set = hash.keys( $ips );
log.info( array.length( $connected_set ) .
          " unique IPs have connected to the site" );
```

See also: hash.values

5.1.22 hash.values(hash)

Returns an array containing the values that have been mapped to in the hash.

Sample Usage

```
# Add up the values of a hash
$values = hash.values( $hash );
$total = 0;
foreach( $val in $values ) {
   $total += $val;
}
log.info( "The total is " . $total );
```

See also: hash.keys

5.1.23 json.deserialize(json_string)

Converts the supplied string in JavaScript Object Notation (JSON) into a TrafficScript array or hash variable. If the supplied string is not in the correct format then a warning will be printed and the result will be empty.

Sample Usage

```
# Deserialising a JSON array
$json_array = '[ "element 1", "element 2" ]';
$array = json.deserialize( $json_array );
# Deserialising a JSON hash
$json_object = '{ "key 1":"value 1", \
                  "key 2":[ "array element" ] }';
$hash = json.deserialize( $json_object );
```

See also: json.serialize

5.1.24 json.serialize(object)

Converts the supplied array or hash variable into JavaScript Object Notation (JSON). This format is commonly used to exchange data between online applications.

Sample Usage

```
# Serialising an array
$array = [ "element 1", "element 2" ];
$json_array = json.serialize( $array );
# Serialising a hash
$hash = [ "key 1" => "value 1",
          "key 2" => [ "array element" ] ];
$json_object = json.serialize( $hash );
```

See also: json.deserialize

5.1.25 lang.assert(condition, message)

If the condition is false, prints a warning to the log with the current line number and terminates the rule. If the condition is true then no messages will be printed and the rule will continue as normal.

Sample Usage

```
# Make sure that this rule is only run with
# Virtual Servers that are using SSL Decryption.
lang.assert( ssl.isSSL(),
             "This rule should only be used \
              with decrypted SSL connections" );
```

See also: lang.warn

5.1.26 lang.chr(number)

Converts a number to the corresponding ASCII character. chr() may be used as an alias for lang.chr().

Sample Usage

```
# Pick a random letter from A - Z
$char = lang.chr( math.random( 26 ) + 65 );
```

Alternative Name: chr

See also: lang.ord

5.1.27 lang.dump(variable)

Converts the supplied variable into a human-readable string. This function is useful for printing the contents of arrays and hashes when debugging TrafficScript rules.

Sample Usage

```
# Print the headers of the current HTTP request
log.info( lang.dump( http.getHeaders() ) );
```

5.1.28 lang.isarray(data)

Returns whether or not the supplied data is an array.

Sample Usage

```
sub passMeAnArray( $array ) {
   # Test whether the supplied data is an array
   if( !lang.isArray( $array ) ) {
      return false;
   }
   # ...
}
```

See also: lang.ishash

5.1.29 lang.ishash(data)

Returns whether or not the supplied data is a hash.

Sample Usage

```
sub passMeAHash( $hash ) {
   # Test whether the supplied data is a hash
   if( !lang.isHash( $hash ) ) {
      return false;
   }
   # ...
}
```

See also: lang.isarray

5.1.30 lang.max(param1, param2)

Returns the maximum value of the two parameters provided. If both parameters are strings, it uses a string comparison; otherwise, the parameters are promoted to integers or doubles and compared. max() may be used as a shorthand for lang.max().

Sample Usage

```
$r = lang.max( 9, 10 ); # returns 10
$s = lang.max( "9", "10" ); # returns "9"
$r = lang.max( 2, "4.8" ); # returns 4.8
```

Alternative Name: max

See also: lang.min

5.1.31 lang.min(param1, param2)

Returns the minimum value of the two parameters provided. If both parameters are strings, it uses a string comparison; otherwise, the parameters are promoted to integers or doubles and compared. min() may be used as a shorthand for lang.min().

Sample Usage

```
$r = lang.min( 9, 10 ); # returns 9
$s = lang.min( "9", "10" ); # returns "10"
$r = lang.min( 2, "4.8" ); # returns 2
```

Alternative Name: min

See also: lang.max

5.1.32 lang.ord(string)

Converts an ascii character to an integer. ord() may be used as a shorthand alias for lang.ord().

Sample Usage

```
# Get the integer value of a character.
$val = lang.ord( "A" );
```

Alternative Name: ord

See also: lang.toString, lang.chr

5.1.33 lang.toArray(values)

Returns an array of the supplied values.

Sample Usage

```
$arr = lang.toArray( "10", "hello" );
```

See also: lang.toDouble, lang.toString, lang.toInt

5.1.34 lang.toDouble(value)

Returns the double (floating point) value of its parameter, using the TrafficScript type-casting rules.

Sample Usage

```
$r = lang.toDouble( "3.14157" ); # returns 3.14157
$r = lang.toDouble( 10 ); # returns 10
$r = lang.toDouble( 3.14175 ); # returns 3.14175
$r = lang.toDouble( "!!!" ); # returns 0.0
```

Alternative Name: toDouble

See also: lang.toInt, lang.toString

5.1.35 lang.toHash(values)

Returns an hash of the supplied key value pairs.

Sample Usage

```
$hash = lang.toHash( "ten", 10, "eleven", 11 );
```

See also: lang.toDouble, lang.toString, lang.toInt, lang.toArray

5.1.36 lang.toInt(value)

Returns the integer value of its parameter, using the TrafficScript type-casting rules.

Sample Usage

```
$r = lang.toInt( "10xxx" ); # returns 10
$r = lang.toInt( 3.14157 ); # returns 3
$r = lang.toInt( 10 ); # returns 10
$r = lang.toInt( "!!!" ); # returns 0
```

Alternative Name: toInt

See also: lang.toDouble, lang.toString

5.1.37 lang.toString(value)

Returns the string value of its parameter, using the TrafficScript type-casting rules.

Sample Usage

```
$s = lang.toString( 10 ); # returns "10"
$s = lang.toString( 3.14157 ); # returns "3.14157"
$s = lang.toString( "10" ); # returns "10"
```

Alternative Name: toString

See also: lang.toInt, lang.toDouble, lang.chr, lang.ord

5.1.38 lang.tochar()

This function is an alias for lang.chr.

Sample Usage

```
# Pick a random letter from A - Z
$char = lang.tochar( math.random( 26 ) + 65 );
```

Alternative Name: tochar

See also: lang.chr

5.1.39 lang.warn(message)

Prints a warning to the log with the line number that this function call appears on. If strict error checking is enabled then the rule will abort.

Sample Usage

```
$decoded = string.decrypt( $password, "passphrase" );
if( !$decoded ) {
   lang.warn( "Failed to decrypt string" );
} else {
   # String decrypt succeeded...
}
```

See also: lang.assert

5.1.40 math.acos(x)

Calculates the arc cosine of x and returns an angle in radians in the range 0 to pi.

Sample Usage

```
$acos = math.acos( 0 ); # returns pi/2 (approx.)
```

Alternative Name: acos

See also: math.cos

5.1.41 math.asin(x)

Calculates the arc sine of x and returns an angle in radians in the range -pi/2 to pi/2.

Sample Usage

```
$asin = math.asin( 0 ); # returns 0 (approx.)
```

Alternative Name: asin

See also: math.sin

5.1.42 math.atan(angle)

Calculates the arc tangent of x and returns an angle in radians in the range -pi/2 to pi/2.

Sample Usage

```
$atan = math.atan( 0 ); # returns 0 (approx.)
```

Alternative Name: atan

See also: math.tan

5.1.43 math.ceil(value)

Returns the smallest integer greater than or equal to its parameter.

Sample Usage

```
$r = math.ceil( 6.28 ); # returns 7
$r = math.ceil( "4" ); # returns 4
```

Alternative Name: ceil

See also: math.floor, math.rint, math.fabs

5.1.44 math.cos(angle)

Interprets its parameter as an angle in radians and returns its cosine.

Sample Usage

```
$cos = math.cos( 6.28314 ); # returns 1 (approx.)
```

Alternative Name: cos

See also: math.sin, math.tan

5.1.45 math.exp(power)

Calculates e raised to the power of its parameter and returns the result.

Sample Usage

```
$r = math.exp( math.ln( 10 ) ); # returns 10
```

Alternative Name: exp

See also: math.ln, math.pow

5.1.46 math.fabs(value)

Interprets its parameter as a floating point number and returns its absolute value.

Sample Usage

```
$r = math.fabs( -6.28 ); # returns 6.28
```

Alternative Name: fabs

See also: math.floor, math.ceil, math.rint

5.1.47 math.floor(value)

Returns the largest integer not greater than its parameter.

Sample Usage

```
$r = math.floor( 4.001 ); # returns 4
$r = math.floor( 17 ); # returns 17
```

Alternative Name: floor

See also: math.ceil, math.rint, math.fabs

5.1.48 math.ln(value)

Returns the natural logarithm of its parameter.

Sample Usage

```
$ln = math.ln( 2.71828 ); # returns 1 (approximately)
```

Alternative Name: ln

See also: math.log, math.exp

5.1.49 math.log(value)

Returns the base10 logarithm of its parameter.

Sample Usage

```
$log = math.log( 100 ); # returns 2
```

Alternative Name: log

See also: math.ln, math.pow

5.1.50 math.pow(num, power)

Raises its first parameter to the power of its second parameter and returns the result.

Sample Usage

```
$r = math.pow( 2, 3 ); # returns 8
```

Alternative Name: pow

See also: math.ln, math.exp, math.sqrt

5.1.51 math.random(range)

Returns a pseudorandom integer greater than or equal to zero, and less than its parameter.

Sample Usage

```
# returns a value in the range 0 to 99
$rand = math.random( 100 );
```

Alternative Name: random

5.1.52 math.rint(value)

Rounds its parameter by returning the integer closest to its value.

Sample Usage

```
$r = math.rint( 4.25 ); # returns 4
$r = math.rint( 4.75 ); # returns 5
```

Alternative Name: rint

See also: math.floor, math.ceil, math.fabs

5.1.53 math.sin(angle)

Interprets its parameter as an angle in radians and returns its sine.

Sample Usage

```
$sin = math.sin( 6.28314 ); # returns 0 (approx.)
```

Alternative Name: sin

See also: math.cos, math.tan

5.1.54 math.sqrt(num)

Returns the square root of its parameter.

Sample Usage

```
$root = math.sqrt( 2 ); # returns 1.414 (approx.)
if( lang.toString( math.sqrt( $num ) ) == "nan" ) {
   # $num is negative or NaN ...
}
```

Alternative Name: sqrt

See also: math.pow

5.1.55 math.tan(angle)

Interprets its parameter as an angle in radians and returns its tangent.

Sample Usage

```
$tan = math.tan( 6.28314 ); # returns 0 (approx.)
```

Alternative Name: tan

See also: math.sin, math.cos

5.1.56 string.BERToInt(string)

Converts a BER compressed integer into an integer.

Sample Usage

```
# $r = 200
$r = string.BERToInt( "\201\110" );
```

Alternative Name: BERToInt

See also: string.intToBER, string.bytesToInt

5.1.57 string.Ireplace(string, search, replacement) - *deprecated*

This function has been deprecated. Use string.replaceI instead.

Replaces the first occurrence of the search string in the supplied string with the replacement. It is case-insensitive and returns the string with the replacement.

5.1.58 string.IreplaceAll(string, search, replacement) - *deprecated*

This function has been deprecated. Use string.replaceAllI instead.

Replaces all occurrences of the search string in the supplied string with the replacement. It is case-insensitive and returns the string with the replacements.

5.1.59 string.append(str1, str2, ...)

Returns the result of concatenating all of its inputs together as strings.

Sample Usage

```
# Returns "The answer is 42"
$s = string.append( "The ", "answer ", "is " , 42 );
```

Alternative Name: append

5.1.60 string.base64decode(string)

Decodes a base64-encoded string and returns the result.

Base64 encoding is used for MIME-encoded messages, and in the HTTP Basic Authorization header.

Sample Usage

```
# Decodes a username and password from HTTP
# BASIC authentication
$h = http.getHeader( "Authorization" );
if( string.startsWith( $h, "Basic " ) ) {
   $enc = string.skip( $h, 6 );
   $userpasswd = string.base64decode( $enc );
   log.info( "User used: ".$userpasswd );
}
```

Alternative Name: base64decode

See also: string.base64encode, string.hexdecode, string.unescape

5.1.61 string.base64encode(string)

Returns the base64-encoded version of the provided string. This converts each group of three characters into a 4-character string containing just [A-Za-z0-9+/], and '=' for padding.

Base64 encoding is used for MIME-encoded messages, and in the HTTP Basic Authorization header.

Sample Usage

```
# Encodes a username and password for HTTP
# BASIC authentication
$enc = string.base64encode( "user:passwd" );
$h = "Basic ".$enc;
http.setHeader( "Authorization", $h );
```

Alternative Name: base64encode

See also: string.base64decode, string.hexencode, string.escape

5.1.62 string.bytesToDotted(string)

Converts a network ordered byte string into an IP address.

Sample Usage

```
# The first 4 bytes are the IP address
$ipstr = string.substring( $msg, 0, 3 );
log.info( "IP is ".string.bytesToDotted( $ipstr ) );
```

Alternative Name: bytesToDotted

See also: string.dottedToBytes, string.bytesToInt

5.1.63 string.bytesToInt(string)

Converts a byte string in network order to an integer. The byte string should be either 1, 2 or 4 bytes long.

Sample Usage

```
# $msg starts with a 2-bytes length
$lenstr = string.substring( $msg, 0, 1 );
$len = string.bytesToInt( $lenstr );
$msg = string.substring( $msg, 2, 2+$len-1 );
```

Alternative Name: bytesToInt

See also: string.intToBytes, string.bytesToDotted, string.BERToInt

5.1.64 string.cmp(str1, str2)

Compares its two parameters as strings in a case-sensitive manner. It returns a negative value if str1 is less than str2; zero if they are equal, and a positive value if str1 is greater than str2.

Sample Usage

```
if( string.cmp( $a, "HTTP/1.0" ) == 0 ) {
   # $a is "HTTP/1.0"
}
if( string.cmp( $a, $b ) < 0 ) {
   # $a is less than $b
}
```

Alternative Name: cmp

See also: string.icmp

5.1.65 string.contains(haystack, needle)

Searches for the provided string (the needle) in the given source (the haystack).

It returns 1 if the 'needle' was found, or 0 otherwise.

Sample Usage

```
if( string.contains( $cookie, "chocolate" ) ) {
   # The cookie contains chocolate ...
}
```

Alternative Name: contains

See also: string.containsI, string.find, string.findr, string.startsWith, string.endsWith

5.1.66 string.containsI(haystack, needle)

Searches for the provided string (the needle) in the given source (the haystack). It is case-insensitive.

It returns 1 if the 'needle' was found, or 0 otherwise.

Sample Usage

```
if( string.containsI( $path, "danger" ) ) {
   # The path contains danger ...
}
```

Alternative Name: containsI

See also: string.contains, string.findI, string.startsWithI, string.endsWithI

5.1.67 string.count(haystack, needle, [start])

Searches from the start of a string, counting the number of times that the provided search string (the needle) is found inside the given string (the haystack). An optional parameter can specify the start position for the search.

It returns the number of times that the string is found.

Sample Usage

```
# Returns 2
$r = string.count( "This is it!", "is" );
# Returns 1, no overlaps allowed
$s = string.count( "ooo", "oo" );
```

Alternative Name: count

See also: string.find, string.contains

5.1.68 string.decrypt(string, passphrase)

Returns the decrypted version of a string that has previously been encrypted using string.encrypt(). The passphrase supplied must match that given to string.encrypt(), otherwise the decoding will fail.

An empty string is returned if the decrypt or the integrity check fails.

Sample Usage

```
# Decrypt the 'kart' cookie
$cookie = http.getcookie( "kart" );
if( $cookie != "" ) {
   $cookie = string.decrypt( $cookie, $passphrase );
   if( $cookie == "" ) {
      log.warn( "User modified kart cookie" );
      http.removecookie( "kart" );
   } else {
      http.setcookie( "kart", $cookie );
   }
}
```

Alternative Name: decrypt

See also: string.encrypt

5.1.69 string.dottedToBytes(IP address)

Converts an IP address to a network order byte string.

Sample Usage

```
# Prepend the client IP onto $msg
$ip = request.getRemoteIP();
$msg = string.dottedToBytes( $ip ).$msg;
```

Alternative Name: dottedToBytes

See also: string.bytesToDotted, string.intToBytes

5.1.70 string.drop(string, count)

Returns all but the last 'count' characters from the end of the provided string. An empty string will be returned if 'count' is greater than the length of the original string.

Sample Usage

```
# returns "www.example"
$s = string.drop( "www.example.com", 4 );
```

Alternative Name: drop

See also: string.skip, string.trim

5.1.71 string.encrypt(string, passphrase)

Encrypts a string using the provided pass phrase. The returned string is encrypted using the AES block cipher, using an expanded form of the passphrase as the cipher key. A MAC is also added to ensure the integrity of the string.

This is open to replay attacks, and as such, should not be used to encrypt sensitive data, such as credit card details.

Sample Usage

```
# Encrypt the 'kart' cookie
$cookie = http.getresponsecookie( "kart" );
if( $cookie != "" ) {
   $cookie = string.encrypt( $cookie, $passphrase );
   http.setresponsecookie( "kart", $cookie );
}
```

Alternative Name: encrypt

See also: string.decrypt

5.1.72 string.endsWith(string, suffix)

Returns 1 if the provided string ends with the given suffix, and 0 otherwise.

Sample Usage

```
if( string.endsWith( $url, ".cgi") ) {
   # Request is for a CGI script ...
}
```

Alternative Name: endsWith

See also: string.endsWithI, string.startsWith, string.contains

5.1.73 string.endsWithI(string, suffix)

Returns 1 if the provided string ends with the given suffix, and 0 otherwise. It is case-insensitive.

Sample Usage

```
if( string.endsWithI( $path, "victory") ) {
   # The path ends with victory
}
```

Alternative Name: endsWithI

See also: string.endsWith, string.startsWithI, string.containsI

5.1.74 string.escape(string)

Returns a percent-encoded version of its parameter.

Control characters and spaces (character value <= 32) and '%' characters are each replaced by a '%' symbol, followed by their 2-digit hex value.

Sample Usage

```
# returns "Hello%20World!%0D%0A"
$s = string.escape( "Hello World!\r\n" );
```

Alternative Name: escape

See also: string.unescape, string.hexencode, string.regexescape, string.urlencode

5.1.75 string.extractHost(string)

Returns the host part of the supplied address if it is a valid IP or hostname. Otherwise the empty string is returned.

Sample Usage

```
sub lookup_proxy( $path ) {
   # Code to map a path to a node
}
# Send certain requests to an external server
$forward = lookup_proxy( http.getPath() );
if( $forward
    && $host = string.extractHost( $forward ) ) {
   if( !$port = string.extractPort( $forward ) ) {
      $port = 80;
   }
   pool.use( "External", $host, $port );
}
```

Alternative Name: extractHost

See also: string.extractPort

5.1.76 string.extractPort(string)

Returns the port part of the supplied address if both the host and port of the supplied address are valid. Otherwise the empty string is returned.

Sample Usage

```
# Get the SIP request's next destination
$next = sip.getRequestHeader( "Route" );
string.regexmatch( $next, "^" );
$next = $1;
# Get the port number of the destination
if( $port = string.extractPort( $next ) ) {
   if( $port < 1024 ) {
      # Don't forward the message
      sip.sendResponse( "403", "Forbidden" );
   }
}
```

Alternative Name: extractPort

See also: string.extractHost

5.1.77 string.find(haystack, needle, [start])

Reports whether the provided search string (the needle) is contained inside the given string (the haystack). An optional parameter can specify the start position for the search.

It returns the location of the first instance of the search string; note that character positions start at 0.

If it could not find the search string, it returns -1.

Sample Usage

```
$r = string.find( "This is it!", "is" ); # Returns 2
```

Alternative Name: find

See also: string.findI, string.findr, string.contains

5.1.78 string.findI(haystack, needle, [start])

Reports whether the provided search string (the needle) is contained inside the given string (the haystack). An optional parameter can specify the start position for the search. It is case-insensitive.

It returns the location of the first instance of the search string; note that character positions start at 0.

If it could not find the search string, it returns -1.

Sample Usage

```
# Returns 0
$r = string.findI( "The way of the warrior", "the" );
```

Alternative Name: findI

See also: string.find, string.containsI

5.1.79 string.findr(haystack, needle, [distanceFromEndToStart])

Searches backwards from the end of a string, determining whether the provided search string (the needle) is contained inside the given string (the haystack). An optional parameter can specify the start position for the search, measured from the end of the string (so setting it to 1 skips the last character in the string).

It returns the location of the last instance of the search string; note that character positions start at 0.

If it could not find the search string, it returns -1.

Sample Usage

```
# Returns 8 (the 'i' in it)
$r = string.findr( "This is it!", "i" );
# Returns 5 (the 'i' in is)
$r = string.findr( "This is it!", "i", 3 );
```

Alternative Name: findr

See also: string.find, string.contains

5.1.80 string.hash(string)

Returns a number representing a hash of the provided string. The returned value should not be relied on to be consistent across different releases of the software.

Sample Usage

```
$hash = string.hash( http.getRawURL() );
connection.setPersistenceKey( $hash % 1000 );
```

Alternative Name: hash

5.1.81 string.hashMD5(string)

Returns the MD5 hash of the provided string. The returned string will be 16 bytes long, and may contain non-printable characters.

Sample Usage

```
$hash = string.hashMD5( $data );
```

Alternative Name: hashMD5

See also: string.hexencode, string.hashSHA1, string.hashSHA256, string.hashSHA384, string.hashSHA512

5.1.82 string.hashSHA1(string)

Returns the SHA1 hash of the provided string. The returned string will be 20 bytes long, and may contain non-printable characters.

Sample Usage

```
$hash = string.hashSHA1( $data );
```

Alternative Name: hashSHA1

See also: string.hexencode, string.hashMD5, string.hashSHA256, string.hashSHA384, string.hashSHA512

5.1.83 string.hashSHA256(string)

Returns the SHA-256 hash of the provided string. The returned string will be 32 bytes long, and may contain non-printable characters.

Sample Usage

```
$hash = string.hashSHA256( $data );
```

Alternative Name: hashSHA256

See also: string.hexencode, string.hashMD5, string.hashSHA1, string.hashSHA384, string.hashSHA512

5.1.84 string.hashSHA384(string)

Returns the SHA-384 hash of the provided string. The returned string will be 48 bytes long, and may contain non-printable characters.

Sample Usage

```
$hash = string.hashSHA384( $data );
```

Alternative Name: hashSHA384

See also: string.hexencode, string.hashMD5, string.hashSHA1, string.hashSHA256, string.hashSHA512

5.1.85 string.hashSHA512(string)

Returns the SHA-512 hash of the provided string. The returned string will be 64 bytes long, and may contain non-printable characters.

Sample Usage

```
$hash = string.hashSHA512( $data );
```

Alternative Name: hashSHA512

See also: string.hexencode, string.hashMD5, string.hashSHA1, string.hashSHA256, string.hashSHA384

5.1.86 string.hexToInt(string)

Converts a hexadecimal number to an integer. Returns the first valid hexadecimal value found in the string, or 0. A prefix of "0x" is accepted, but not required. Negative numbers are also valid.

Sample Usage

```
# Returns 10.
$int = string.hexToInt( "0000a" );
```

See also: string.intToHex

5.1.87 string.hexdecode(encoded string)

Returns the hex-decoded version of the provided string. This interprets each character pair as a 2-digit hex value, replacing it with the corresponding 8-bit character. It does not verify that the original string was correctly encoded.

Sample Usage

```
# returns "hello"
$s = string.hexdecode("68656C6C6F");
```

Alternative Name: hexdecode

See also: string.hexencode, string.base64decode, string.unescape

5.1.88 string.hexencode(string)

Returns the hex-encoded version of the provided string . This converts each character into a two-character hex representation, doubling the length of the string.

Sample Usage

```
# Returns "68656C6C6F"
$s = string.hexencode( "hello" );
```

Alternative Name: hexencode

See also: string.hexdecode, string.base64encode, string.escape

5.1.89 string.htmldecode(encodedstring)

Returned the HTML-decoded version of a string, converting symbols such as < and "

Sample Usage

```
$body = string.htmldecode( http.getBody( 0 ) );
```

Alternative Name: htmldecode

See also: string.htmlencode

5.1.90 string.htmlencode(string)

Returns the encoded version of the supplied string to make it safe for including in HTML. It converts '<' to < '>' to >, '"' to " and '&' to &. Control characters are hex-encoded.

Sample Usage

```
$html = string.htmlencode( $parameter );
```

Alternative Name: htmlencode

See also: string.htmldecode, string.urlencode

5.1.91 string.icmp(str1, str2)

Compares its two parameters as strings in a case-insensitive manner. It returns a negative value if str1 is less than str2; zero if they are equal, and a positive value if str1 is greater than str2.

Sample Usage

```
if( string.icmp( $a, "Content-Length" ) == 0 ) {
   # $a is "Content-Length"
}
```

Alternative Name: icmp

See also: string.cmp

5.1.92 string.insertBytes(string, insertion, offset)

Inserts a string into another string at the supplied offset, and returns the resulting string. If offset < 0, or offset > length(string), the original string is returned unchanged. If offset == 0 the insertion string is prepended; if offset == length(string) the insertion string is appended.

Sample Usage

```
# $r = "he was Othello"
$r = string.insertbytes( "hello", " was Othe", 2 );
# $r = "hello world"
$r = string.insertbytes( "hello", " world", 5 );
# $r = "I say hello"
$r = string.insertbytes( "hello", "I say ", 0 );
```

Alternative Name: insertBytes

See also: string.replaceBytes

5.1.93 string.intToBER(number)

Converts an integer into a BER compressed integer (which is a variable-length binary string encoding).

Sample Usage

```
# $r = "\201\110"
$r = string.intToBER( 200 );
```

Alternative Name: intToBER

See also: string.BERToInt, string.intToBytes

5.1.94 string.intToBytes(number, [width])

Converts an integer to a network order byte string of the specified width. Only widths of 1, 2 and 4 are permitted, and the width defaults to 4 if it is not supplied.

Sample Usage

```
# Prepend $msg with its length, as a 4-byte int
string.insertBytes( $msg,
   string.intToBytes( string.len( $msg ) ),
   0 );
```

Alternative Name: intToBytes

See also: string.bytesToInt, string.bytesToDotted, string.intToBER

5.1.95 string.intToHex(string)

Converts an integer into a hexadecimal string.

Sample Usage

```
# Returns "0000cafe"
$hex = string.intToHex( 51966 );
```

See also: string.hexToInt

5.1.96 string.ipmaskmatch(IP Address, CIDR IP Subnet)

Returns 1 if the provided IP address is contained in the CIDR IP Subnet, and 0 otherwise.

It interprets its first parameter as a string containing an IP address, and its second parameter as an CIDR IP subnet. CIDR IP subnets can be of the form "10.0.1.0/24", "10.0.1.0/255.255.255.0", "10.0.1." or "10.0.1.1".

For IPv6, the standard notation of "2001:200:0:8002::/64" is supported.

Sample Usage

```
if( string.ipmaskmatch( $ip, "10.0.0.0/8" ) ) {
   # IP is in the 10.*.*.* subnet ...
}
```

Alternative Name: ipmaskmatch

See also: string.validIPAddress

5.1.97 string.left(string, count)

Returns the first 'count' characters of the provided string. An empty string will be returned if 'count' is less than or equal to zero.

Sample Usage

```
# returns "#!"
$s = string.left( "#!/bin/sh", 2 );
```

Alternative Name: left

See also: string.skip

5.1.98 string.len(string)

Interprets its parameter as a string and returns its length (in bytes).

Sample Usage

```
$len = string.len( "Hello world!" ); # returns 12
```

Alternative Name: len

5.1.99 string.length(string)

This function is an alias for string.len.

Sample Usage

```
$len = string.length( "Hello world!" ); # returns 12
```

Alternative Name: length

See also: string.len

5.1.100 string.lowercase(string)

Converts all characters in the provided string to lowercase and returns the result.

Sample Usage

```
$s = string.lowercase("AbCdEfG"); # Returns "abcdefg"
```

Alternative Name: lowercase

See also: string.uppercase

5.1.101 string.normalizeIPAddress(string)

Returns a unique string representation of an IP address: all leading zeros are removed, and for IPv6 addresses the first occurrence of blocks consisting entirely of zeros is replaced by "::". This normal form can be used to compare IP addresses without ambiguity and is also the form used by TrafficScript functions returning IP addresses. If the string is not a valid IP address, the empty string is returned.

Sample Usage

```
$remote_ip = request.getremoteip();
$client = string.normalizeipaddress(
            "2001:0:0157::0:006a" );
# $client is now "2001::157:0:0:0:0:6a"
if( $remote_ip == $client ){
   # do something specific for this client
}
```

Alternative Name: normalizeIPAddress

See also: string.validIPAddress

5.1.102 string.randomBytes(length)

Returns a string of the supplied length filled with random bytes.

Sample Usage

```
# Get a string 16 bytes long filled with random data
$str = string.randomBytes( 16 );
```

Alternative Name: randomBytes

5.1.103 string.regexescape(string)

Returns a version of its parameter suitable for using inside a regex match as a string literal.

All characters in the string that aren't a-z, A-Z, 0-9 or '_' are escaped using a backslash.

Sample Usage

```
$str = "10.100.230.5";
$escaped = string.regexescape( $str );
if( string.regexmatch( $line, $escaped ) ) {
   log.info( "Line matched: " . $str );
}
```

Alternative Name: regexescape

See also: string.escape

5.1.104 string.regexmatch(string, regex, [flags]))

Performs a regular expression match on the supplied string. If the regular expression 'regex' contains bracketed sub-expressions, then the variables $1 ... $9 will be set to the matching substrings.

Note that the '\' character is an escape character in TrafficScript strings enclosed with double quotation marks. If you need to put a literal '\' in a regular expression, you must escape it as '\\' or enclose the string in single quotation marks. To match a literal string inside a regular expression use the string.regexEscape function.

The optional 'flags' parameter contains a string of single-letter options. The following options are supported:

- 'i', meaning 'case insensitive' - letters in the pattern match both upper and lower case letters.

It returns 1 if matched, and 0 otherwise.

Sample Usage

```
$id = "user=Joe, password=Secret";
if( string.regexmatch(
   $id, "^user=(.*), password=(.*)$" ) ) {
   log.info( "Got UserID: ".$1.", Password: ".$2 );
}
$name = "Name( Bob )";
string.regexmatch( $name, 'Name\( (\S+) \)' );
log.info( $1 ); # prints Bob
```

Alternative Name: regexmatch

See also: string.wildmatch, string.regexsub, string.regexescape

5.1.105 string.regexsub(string, regex, replacement, [flags])

Performs a regular expression match on the supplied string, then replaces each matching substring with the supplied replacement. The replacement string may contain $1 .. $9 substitutions, which reference bracketed sub-expressions in the regular expression.

Note that the '\' character is an escape character in TrafficScript strings enclosed with double quotation marks. If you need to put a literal '\' in a regular expression, you must escape it as '\\' or enclose the string in single quotation marks. To match a literal string inside a regular expression use the string.regexEscape function.

The optional 'flags' parameter contains a string of single-letter options. The following options are supported:

- 'g', meaning 'global replace' - apply the regex pattern as many times as possible.
- 'i', meaning 'case insensitive' - letters in the pattern match both upper and lower case letters.

string.regexsub() returns the string with the replacements.

Sample Usage

```
# Rewrite incoming URL
$url = string.regexsub($url, "^/(.*)/secure",
                       "/$1/private");
# Remove cookieless session from URL
# e.g. "/(sessionid)/app/index" => "/app/index"
$url = string.regexsub( $url, '/\(\S+\)/', "/" );
```

Alternative Name: regexsub

See also: string.regexmatch

5.1.106 string.replace(string, search, replacement)

Replaces the first occurrence of the search string in the supplied string with the replacement. It is case-sensitive and returns the string with the replacement.

Sample Usage

```
# Rewrite incoming URL
$url = string.replace($url, "/secure", "/private");
```

Alternative Name: replace

See also: string.replaceI, string.replaceAll, string.replaceAllI

5.1.107 string.replaceAll(string, search, replacement)

Replaces all occurrences of the search string in the supplied string with the replacement. It is case-sensitive and returns the string with the replacements.

Sample Usage

```
# Rewrite incoming URL
$url = string.replaceAll($url, "/in", "/out");
```

Alternative Name: replaceAll

See also: string.replaceAllI, string.replace, string.replaceI

5.1.108 string.replaceAllI(string, search, replacement)

Replaces all occurrences of the search string in the supplied string with the replacement. It is case-insensitive and returns the string with the replacements.

Sample Usage

```
# Rewrite incoming URL
$url = string.replaceAllI($url, "/in", "/out");
```

Alternative Name: replaceAllI

See also: string.replaceAll, string.replaceI, string.replace

5.1.109 string.replaceBytes(string, replacement, offset)

Replaces a portion of a string with a replacement string at the supplied offset. It returns a modified version of the original string of the same length as the original with the appropriate bytes replaced from the replacement string.

If offset < 0, or offset > length(string), or length(replacement) == 0, string.replaceBytes returns the original string.

Sample Usage

```
# $r = "hi lo"
$r = string.replaceBytes( "hello", "i ", 1 );
# $r = "helwo"
$r = string.replaceBytes( "hello", "world", 3 );
```

Alternative Name: replaceBytes

See also: string.insertBytes

5.1.110 string.replaceI(string, search, replacement)

Replaces the first occurrence of the search string in the supplied string with the replacement. It is case-insensitive and returns the string with the replacement.

Sample Usage

```
# Rewrite incoming URL
$url = string.replaceI($url, "/secure", "/private");
```

Alternative Name: replaceI

See also: string.replace, string.replaceAllI, string.replaceAll

5.1.111 string.reverse(string)

Returns the characters of a string in reverse order.

Sample Usage

```
$c = string.reverse( "esrever" ); # Returns "reverse"
```

Alternative Name: reverse

5.1.112 string.right(string, count)

Returns the last 'count' characters of the provided string. An empty string will be returned if 'count' is less than or equal to zero.

Sample Usage

```
# returns "sh"
$s = string.right( "#!/bin/sh", 2 );
```

Alternative Name: right

See also: string.drop

5.1.113 string.skip(string, count)

Returns all but the first 'count' characters from the input string. An empty string will be returned if 'count' is greater than the length of the original string.

Sample Usage

```
# returns "/bin/sh"
$s = string.skip( "#!/bin/sh", 2 );
```

Alternative Name: skip

See also: string.drop, string.trim

5.1.114 string.split(string, [separator])

Returns an array containing the substrings of the supplied string that are delimited by the separator. The separator defaults to a space character, so applying this method to a string without supplying the separator character will return an array containing all the individual words in the string.

Sample Usage

```
$words = string.split( http.getResponseBody() );
log.info( "There were " . array.length( $words )
          . " words in the response" );
```

See also: array.join

5.1.115 string.sprintf(format string, arguments)

Behaves like the C library sprintf() function. Only %s, %c, %d and %f are supported. The function returns the generated string.

Sample Usage

```
$text = string.sprintf(
   "Apples: %3d Oranges: %3d\n",
   $apples, $oranges );
```

Alternative Name: sprintf

See also: string.append

5.1.116 string.startsWith(string, prefix)

Returns 1 if the provided string starts with the given prefix, and 0 otherwise.

Sample Usage

```
if( string.startsWith( $url, "http://" ) ) {
   # URL is of the form http://host/uri ...
}
```

Alternative Name: startsWith

See also: string.startsWithI, string.endsWith, string.contains

5.1.117 string.startsWithI(string, prefix)

Returns 1 if the provided string starts with the given prefix, and 0 otherwise. It is case-insensitive.

Sample Usage

```
if( string.startsWithI( $path, "/tea" ) ) {
   # The path starts with tea ...
}
```

Alternative Name: startsWithI

See also: string.startsWith, string.endsWithI, string.containsI

5.1.118 string.substring(string, base, end)

Returns the substring starting at character position 'base' and ending at position 'end'.

Note that character positions start at 0, and the end position is inclusive.

Sample Usage

```
# Set $s to "is is a"
$s = string.substring( "This is a string", 2, 8 );
```

Alternative Name: substring

5.1.119 string.trim(string)

Returns the result of removing leading and trailing white space (and control characters) from its input

Sample Usage

```
$s = string.trim( " 1234 " ); # returns "1234"
```

Alternative Name: trim

See also: string.skip, string.drop

5.1.120 string.unescape(escaped string)

Returns the escape-decoded version of its parameter.

%-encoded characters are replaced with their decoded versions. %u-encoded characters are replaced with their UTF-8 representation. If there are illegal digits which cannot safely be converted, the variable $1 is set to 0 and the result contains '_' in place of the '%'. Such malicious %-escaped URLs are a common way of attacking unassuming servers or applications, and by handling them in this way, the attack is thwarted, but some information about a suspicious request is retained.

Sample Usage

```
# returns "\r\n100%"
$s = string.unescape("%0D%0A100%25");
# returns "something_BGelse"
$s = string.unescape("something%BGelse");
# returns "file.ida"
$s = string.unescape("file.%u0069%u0064%u0061");
```

Alternative Name: unescape

See also: string.escape, string.hexdecode

5.1.121 string.uppercase(string)

Converts all characters in the provided string to uppercase and returns the result.

Sample Usage

```
$s = string.uppercase("aBcDeFg"); # Returns "ABCDEFG"
```

Alternative Name: uppercase

See also: string.lowercase

5.1.122 string.urlencode(string)

Returns the URL-encoded version of the supplied string to make it safe for including in URLs. It converts unsafe characters to percent+hex form.

Sample Usage

```
$url = string.urlencode( $parameter );
```

Alternative Name: urlencode

See also: string.unescape, string.htmlencode

5.1.123 string.validIPAddress(string)

Returns 4 if the string provided is an IPv4 address, 6 if it is an IPv6 address and 0 if it is not a valid IP address.

Sample Usage

```
$ipversion = string.validIPAddress( $x );
if( 4 == $ipversion ) {
   # do something specific to IPv4
} else if( 6 == $ipversion ) {
   # do something specific to IPv6
} else {
   # error: not a valid IP address
}
```

Alternative Name: validIPAddress

See also: string.ipmaskmatch

5.1.124 string.wildmatch(string, pattern)

Performs a shell-like wild match on the supplied string. The pattern may contain the wildcard characters '?' (which matches a single character) and '*' (which matches any substring).

It returns 1 if matched, and 0 otherwise.

Sample Usage

```
$url = http.getPath();
if( string.wildmatch( $url, "*.cgi" ) ) {
   # Is a request for a CGI script ...
}
```

Alternative Name: wildmatch

See also: string.regexmatch

5.1.125 string.gmtime.parse(str)

Parses the supplied string as a time stamp and returns the time in seconds since the epoch (1st Jan 1970). Dates before the epoch or after 2038 will produce unexpected results.

Note: Timezone information contained inside the time string is ignored. The time is always assumed to be in GMT.

Sample Usage

```
# Parse a string into unix time format
$str = "Tue, 21 Oct 2008 13:44:26 GMT";
$time = string.gmtime.parse( $str );
```

See also: sys.gmtime.format

5.1.126 sys.domainname()

Returns the domain name of the host machine. For example, if the machine is named "server1.example.com", sys.domainname() will return "example.com".

Sample Usage

```
$domainname = sys.domainname();
```

Alternative Name: domainname

See also: sys.hostname

5.1.127 sys.getenv(variable)

Returns the named environment variable, or the empty string if the environment variable does not exists.

Sample Usage

```
$zeushome = sys.getenv( "ZEUSHOME" );
```

Alternative Name: getenv

5.1.128 sys.getpid()

Returns the process id of the current process.

Sample Usage

```
$mypid = sys.getpid();
```

Alternative Name: getpid

5.1.129 sys.hostname()

Returns the hostname of the host machine. For example, if the machine is named "server1.example.com", sys.hostname() will return "server1".

Sample Usage

```
$hostname = sys.hostname();
```

Alternative Name: hostname

See also: sys.domainname

5.1.130 sys.time()

Returns the current system time as the number of seconds since midnight, 1/1/1970.

Sample Usage

```
$unixtime = sys.time();
```

Alternative Name: time

See also: sys.timeToString, sys.localtime.format, sys.gmtime.format, sys.time.highres

5.1.131 sys.timeToString(unixtime)

Takes the time in seconds since midnight, 1/1/1970 and if the optional unixtime parameter is provided, returns a formatted string representing that time. If the unixtime parameter is not given, it returns the current time as a formatted string.

Sample Usage

```
# Returns "[01/Feb/2004:12:24:51 +2000]"
$tm = sys.timeToString( sys.time() );
```

Alternative Name: timeToString

See also: sys.time

5.1.132 sys.gmtime.format(format, unixtime)

Converts the time into a string format. This function converts using GM time - see sys.localtime.format() to convert using localtime.

Format	Meaning	Format	Meaning
%a	Mon Tue Wed ...	%A	Monday Tuesday ...
%b	Jan Feb Mar ...	%B	January February ...
%d	Day of month "01"-"31"	%D	%m/%d/%y
%e	Day of month " 1"-"31"	%H	Hour of day "00-23"
%h	Equivalent to %b	%I	Hour of day "01" - "12"
%j	Julian day of the year "001" to "366"	%m	Month of year "01" - "12"
%M	Minute "00" - "59"	%n	Newline character
%p	AM/PM	%r	Time in %I:%M:%S [AM|PM]
%R	%H:%M	%S	Seconds, output as a number between "00" and "61"
%t	Tab character	%T	%H:%M:%S
%u	Day of week (1 = Monday, 7 = Sunday)	%w	Day of week (0 = Sunday, 6 = Saturday)
%y	Year (without century) "00" to "99"	%Y	Year "0000" to "9999"
%Z	Time zone ("GMT")	%%	"%"

If supplied, the optional 'unixtime' parameter specifies the number of seconds since midnight 1/1/1970, and the function returns a formatted string representing that time. If the 'unixtime' value is not provided, the current time will be returned.

Sample Usage

```
# Return a time suitable for an HTTP header
# e.g. Mon, 14 Aug 2006 12:39:01 GMT
$str = sys.gmtime.format( "%a, %d %b %Y %T GMT" );
```

Alternative Name: gmtime.format

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.month, sys.time.year, sys.time.yearday, sys.localtime.format, string.gmtime.parse

5.1.133 sys.localtime.format(format, unixtime)

Converts the time into a string format. This function converts using localtime - see sys.gmtime.format() to convert using GMT.

Format	Meaning	Format	Meaning
%a	Mon Tue Wed ...	%A	Monday Tuesday ...
%b	Jan Feb Mar ...	%B	January February ...
%d	Day of month "01"-"31"	%D	%m/%d/%y
%e	Day of month " 1"-"31"	%H	Hour of day "00-23"
%h	Equivalent to %b	%I	Hour of day "01" - "12"
%j	Julian day of the year "001" to "366"	%m	Month of year "01" - "12"
%M	Minute "00" - "59"	%n	Newline character
%p	AM/PM	%r	Time in %I:%M:%S [AM|PM]
%R	%H:%M	%S	Seconds, output as a number between "00" and "61"
%t	Tab character	%T	%H:%M:%S
%u	Day of week (1 = Monday, 7 = Sunday)	%w	Day of week (0 = Sunday, 6 = Saturday)
%y	Year (without century) "00" to "99"	%Y	Year "0000" to "9999"
%Z	Time zone (from $TZ)	%%	"%"

If supplied, the optional 'unixtime' parameter specifies the number of seconds since midnight 1/1/1970, and the function returns a formatted string representing that time. If the 'unixtime' value is not provided, the current time will be returned.

Sample Usage

```
# Return a formatted string
# e.g. Mon, 14 Aug 2006 12:39:01 EST
$str = sys.localtime.format( "%a, %d %b %Y %T EST" );
```

Alternative Name: localtime.format

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.month, sys.time.year, sys.time.yearday, sys.gmtime.format

5.1.134 sys.time.highres()

Returns the current system time as the number of seconds and microseconds since midnight, 1/1/1970. The value is returned as a double, e.g. 1138417190.823265

Sample Usage

```
$time = sys.time.highres();
```

Alternative Name: time.highres

See also: sys.timeToString, sys.localtime.format, sys.gmtime.format, sys.time

5.1.135 sys.time.hour(unixtime)

Returns the hour of the day in local time (0-23).

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$hour = sys.time.hour();
```

Alternative Name: time.hour

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.weekday, sys.time.monthday, sys.time.yearday, sys.time.month, sys.time.year, sys.localtime.format

5.1.136 sys.time.minutes(unixtime)

Returns the minutes after the hour in local time (0-59).

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$mins = sys.time.minutes();
```

Alternative Name: time.minutes

See also: sys.time, sys.time.seconds, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.yearday, sys.time.month, sys.time.year, sys.localtime.format

5.1.137 sys.time.month()

Returns the month of the year in local time (1-12).

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
# Find out what the month is tomorrow.
$month = sys.time.month( sys.time() + 86400);
```

Alternative Name: time.month

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.yearday, sys.time.year, sys.localtime.format

5.1.138 sys.time.monthday(unixtime)

Returns the day of the month in local time (1-31).

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$dayofmonth = sys.time.monthday();
```

Alternative Name: time.monthday

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.yearday, sys.time.month, sys.time.year, sys.localtime.format

5.1.139 sys.time.seconds(unixtime)

Returns the seconds after the minute in local time. Normally, it returns a number in the range of (0-59), but can be up to 61 to allow for leap seconds.

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$secs = sys.time.seconds();
```

Alternative Name: time.seconds

See also: sys.time, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.yearday, sys.time.month, sys.time.year, sys.localtime.format

5.1.140 sys.time.weekday(unixtime)

Returns the day of the week in local time (1-7). Sunday has the value 1; Saturday has the value 7.

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$dayofweek = sys.time.weekday();
```

Alternative Name: time.weekday

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.monthday, sys.time.yearday, sys.time.month, sys.time.year, sys.localtime.format

5.1.141 sys.time.year(unixtime)

Returns the year in local time (1970-2038).

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$year = sys.time.year();
```

Alternative Name: time.year

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.yearday, sys.time.month, sys.localtime.format

5.1.142 sys.time.yearday(unixtime)

Returns the day of the year in local time (1-366).

If optional parameter 'unixtime' is supplied, then this specifies the time since midnight 1/1/1970 otherwise the current time will be used.

Sample Usage

```
$dayofyear = sys.time.yearday();
```

Alternative Name: time.yearday

See also: sys.time, sys.time.seconds, sys.time.minutes, sys.time.hour, sys.time.weekday, sys.time.monthday, sys.time.month, sys.time.year, sys.localtime.format

5.2 Zeus Traffic Manager Functions

Zeus TrafficScript functions provide the means to inspect, manipulate and direct traffic within Zeus Traffic Manager.

The TrafficScript functions are grouped into several families:

- **connection.**: These low-level functions allow you to query and manipulate the connection directly;

- **connection.data.**: These functions allow you to set and query connection-local data;
- **counter.**: Provides a simple counter increment mechanism;
- **data.**: These functions allow you to store and retrieve persistent variables;
- **event.**: Provides the ability to trigger a custom event;
- **geo.**: These functions provide various geo-location abilities to the **Zeus Multi-Site Manager** product;
- **http.**: These helper functions allow you to query and manipulate HTTP connections easily, without having to parse and interpret the connection directly;
- **http.cache.**: These functions give you control over the Web Cache used for caching HTTP content;
- **http.compress.**: These functions allow you to control whether or not compressible responses are compressed;
- **http.request.**: These functions can be used to issue HTTP GET or POST requests, and return the result;
- **http.stream.**: These functions can be used to manipulate streaming data over HTTP;
- **java.**: Runs a named Java Extension;
- **log.**: These functions can be used to append messages to the error log file;
- **net.dns.**: These functions can be used to perform DNS lookups in a TrafficScript rule;
- **pool.**: These functions are used to select the pool to balance the connection with;
- **rate.**: Use these functions to select a rate shaping class;
- **request.**: These low-level functions allow you to query and manipulate the client-side of the connection directly;
- **resource.**: These functions query external resources;
- **response.**: These low-level functions allow you to query and manipulate the serverside of the connection directly;
- **rtsp.**: These functions provide control over Real Time Streaming Protocol (RTSP) traffic;
- **rule.**: These functions can examine the current rule processing situation;
- **sip.**: These functions provide control over Session Initiation Protocol (SIP) traffic;
- **slm.**: These functions are used to query and assign service level monitoring properties to a connection.

- **ssl.**: These functions are used to query the SSL parameters of encrypted connections;
- **xml.**: These functions are used to query and manipulate XML data.

5.2.1 connection.checkLimits([poolname])

This function checks to see if the connection will be queued due to backend connection limits. The function returns 1 if the connection is within configured maximum limits for the named pool (See max_connections_pernode setting in Pool > Connection Management). The function returns 0 if the connection will exceed the configured maximum limits and will be queued.

If the named pool does not exist, your traffic manager will log a warning message and a value of -1 will be returned.

Sample Usage

```
# A connection is queued if it exceeds
# max_connections_pernode.
# Send a failure response to the client.
if( !connection.checkLimits( "pool" ) ) {
   http.sendResponse( "500 Failure",
                      "text/plain",
                      "Reached maximum connections",
                      "" );
   connection.discard();
}
```

5.2.2 connection.close(Data, [Read])

Writes the provided data directly back to the client. After the data has been sent, the connection is closed.

The optional second argument specifies whether data should continue to be read in from the client after sending this response, and wait for it to close the connection. If set to 0, the connection will close immediately. If non-zero, the traffic manager will wait and read any remaining data from the connection before closing it.

The default behaviour is to wait, because some client software will not read a response until it has sent its entire request.

Sample Usage

```
# Send an instant response and close the connection
connection.close( "500 Go away\r\n" );
```

See also: connection.discard, http.sendResponse

5.2.3 connection.discard()

Immediately closes the current connection and stops processing rules. This is equivalent to the function call 'pool.use("discard")'.

Sample Usage

```
# Drop this connection NOW!
connection.discard();
```

See also: connection.close, http.sendResponse

5.2.4 connection.getBandwidthClass() - *deprecated*

This function has been deprecated. Use response.getBandwidthClass instead.

Returns the current bandwidth class for the connection to the client, or an empty string if no class is set.

5.2.5 connection.getData(count) - *deprecated*

This function has been deprecated. Use request.get instead.

Returns the first 'count' bytes of data provided by the client.

Warning: you can stall a connection by asking it to read more data than the remote client will provide. Combine this with connection.getDataLen() to reliably read data from a connection.

5.2.6 connection.getDataLen() - *deprecated*

This function has been deprecated. Use request.getLength instead.

Returns the number of bytes of data already received from the client. This can be combined with connection.getData() to reliably read data from a connection without stalling if no data is available.

5.2.7 connection.getLine(offset) - *deprecated*

This function has been deprecated. Use request.getLine instead.

Returns a line of input data provided by the client. The line separator is '\n', and this is stripped off before returning the line. connection.getline() takes a single byte-count argument which indicates where to start scanning for a line; a value of '0' begins at the start, so returns the first line.

When connection.getline() returns, the variable $1 is updated to point to the start of the next line in the datastream.

You can iterate through the lines of input data by using $1 as the iterator variable.

5.2.8 connection.getLocalIP() - *deprecated*

This function has been deprecated. Use request.getLocalIP instead.

Returns the IP address that the client connected to, i.e. the address local to this machine.

5.2.9 connection.getLocalPort() - *deprecated*

This function has been deprecated. Use request.getLocalPort instead.

Returns the network port number that the client connected to. (e.g. port 80 is normal for a web server)

5.2.10 connection.getMemoryUsage()

Returns an estimate of the amount of memory currently in use for this connection, in bytes. Memory is primarily used for buffering data, and the memory usage can be tuned using the various buffer size settings.

Sample Usage

```
# How much memory are we using?
$memoryusage = connection.getMemoryUsage();
```

5.2.11 connection.getNode()

Returns the name of the back-end node that this request is connected to. If a back-end node has not been chosen, which is normally the case in request rules, it returns the empty string.

Sample Usage

```
# Which node is used for this connection
$nodename = connection.getNode();
```

See also: connection.getPool, connection.getVirtualServer, request.avoidNode

5.2.12 connection.getPersistence()

In a Response rule this function returns the name of the current Session Persistence class used for this connection, or whatever class has been set by a previous use of `connection.setPersistence()`.

Sample Usage

```
$class = connection.getPersistence();
```

See also: connection.setPersistence

5.2.13 connection.getPool()

Returns the name of the pool that this request is connected to. If a pool has not been chosen, it returns the empty string.

Sample Usage

```
# Where are we connected to?
$poolname = connection.getPool();
```

See also: connection.getNode, connection.getVirtualServer

5.2.14 connection.getRemoteIP() - *deprecated*

This function has been deprecated. Use request.getRemoteIP instead.

Returns the remote IP address of the client.

5.2.15 connection.getRemotePort() - *deprecated*

This function has been deprecated. Use request.getRemotePort instead.

Returns the remote port of the client's connection.

5.2.16 connection.getServiceLevelClass()

Returns the current service level class for the connection, or an empty string if no class is set.

Sample Usage

```
$class = connection.getServiceLevelClass();
```

See also: connection.setServiceLevelClass

5.2.17 connection.getVirtualServer()

Returns the name of the Virtual Server that the rule is running under.

Sample Usage

```
# Are we on the secure site?
if( connection.getVirtualServer() == "secure" ) {
   pool.use( "secure" );
}
```

See also: connection.getNode, connection.getPool

5.2.18 connection.setBandwidthClass() - *deprecated*

This function has been deprecated. Use response.setBandwidthClass instead.

Sets the bandwidth class for the current connection to the client. Providing an empty class name removes the bandwidth class from the connection. It returns zero if an error occurs (for example, if the bandwidth class does not exist), and 1 otherwise.

5.2.19 connection.setData(request data) - *deprecated*

This function has been deprecated. Use request.set instead.

Replaces the input data read from the client with the provided string.

This is a low-level routine that should be used with care. For protocols with their own higher-level protocol managers (e.g. HTTP), use the higher level routines to alter the input data (e.g. http.setHeader() and http.setBody()).

5.2.20 connection.setIdempotent(resend) - *deprecated*

This function has been deprecated. Use request.setIdempotent instead.

Marks a request as resendable or non-resendable.

An *idempotent* request has no detrimental side effects, so it can safely be attempted multiple times. A non-idempotent request has a side effect - for example, it may update a database, or initiate a purchase.

By default, all non-HTTP requests are marked as idempotent. If a back-end node fails to generate a correct response when a request is initially forwarded to it, your traffic manager will attempt to resend the request to another node.

connection.setIdempotent() can override this behaviour. If 'resend' is zero, this indicates that the request should only be attempted against one back-end node.

If 'resend' has a non-zero value, this indicates that if a request is made to a back-end node and a correct response is not received, your traffic manager should retry the request against another back-end node.

Note that a request cannot be resent once it has begun streaming data between the client and the node. Additionally, UDP connections cannot be marked as resendable (the UDP client application should handle failed UDP responses).

5.2.21 connection.setPersistence(name)

Sets the Session Persistence class that will be used for the connection. This is used to override the default Session Persistence class that will be used once a Pool is selected.

If no parameter is given then the current Session Persistence class will be removed and the Pool's default session persistence class will be used for this connection.

Sample Usage

```
connection.setPersistence( "sales" );
```

See also: connection.getPersistence, connection.setPersistenceKey

5.2.22 connection.setPersistenceKey(value)

Sets the value of the Session Persistence key that is used by a Universal Session Persistence type class.

Setting the value to the empty string will remove any persistence key from the connection.

A Session Persistence class that uses Universal Session Persistence attempts to ensure that every connection that provides the same key is directed to the same back-end node.

This function has no effect if a different type of session persistence class is ultimately used.

Sample Usage

```
$value = http.getHeader( "User-Agent" ) .
         request.getRemoteIP();
connection.setPersistenceKey( $value );
connection.setPersistence( "my persistence class" );
```

See also: connection.setPersistence

5.2.23 connection.setPersistenceNode(value)

Sets the back-end node to be used by a NamedNode Persistence class.

A Session Persistence class that uses NamedNode Persistence will then ensure that this node will be used for the request. The node must be valid and exist in the Pool being used. If no port number is given, or if the port number is not valid, then if there is a node with a matching name, it will be used. For example, if the node 'web:80' is specified, but there is only a 'web:443', then that node will be used instead. This is to help share session persistence between different services on the same machine.

This function has no effect if a different type of session persistence class is ultimately used.

Sample Usage

```
# Use the node 'web:80' for this request
connection.setPersistenceNode( "web:80" );
```

See also: connection.setPersistence

5.2.24 connection.setServiceLevelClass()

Sets the service level class for the current connection. It returns zero if an error occurs (for example, if the service level class does not exist), and 1 otherwise

Sample Usage

```
connection.setServiceLevelClass( "gold" );
```

See also: connection.getServiceLevelClass

5.2.25 connection.sleep(milliseconds)

Pauses processing of the current connection for the specified number of milliseconds. This can be used to rate-limit particular clients; for example, those asking for particular files, or from particular locations, or using particular user-agents.

Sample Usage

```
# Pause this connection for 2 seconds
connection.sleep( 2000 );
```

5.2.26 connection.data.get(key)

Returns the value that was previously stored with the given key using connection.data.set() in the current connection, or returns the empty string if no data was stored.

Sample Usage

```
# Track the state of this connection so that we
# can process the response correctly
$req = request.get( 5 );
if( string.startsWith( $req, "LOGIN" ) ) {
   connection.data.set( "state", "login" );
} else {
   connection.data.set( "state", "" );
}
# ...
# in the response, ...
$state = connection.data.get( "state" );
```

See also: connection.data.set, data.get

5.2.27 connection.data.set(key, value)

Stores a value for this connection, associating it with the provided key. The value can be retrieved later when processing the same connection, using connection.data.get(). Once the connection finishes, the value cannot be retrieved.

Sample Usage

```
# Track the state of this connection so that we
# can process the response correctly
$req = request.get( 5 );
if( string.startsWith( $req, "LOGIN" ) ) {
   connection.data.set( "state", "login" );
} else {
   connection.data.set( "state", "" );
}
# ...
# in the response, ...
$state = connection.data.get( "state" );
```

See also: connection.data.get, data.set

5.2.28 counter.increment(counter, [amount])

Increments the numbered counter. These counters are readable via SNMP, and can be graphed on the Current Activity page on the Administration Server.

By default, the counter is incremented by one, but you can also supply an amount to increment the counter by. If a negative amount is supplied the counter is decremented.

Sample Usage

```
# Increment the first user counter
counter.increment( 1 );
```

5.2.29 data.get(key)

Returns the value that was previously stored with the given key using data.set(), or returns the empty string if no data was stored.

Values stored in this way are persistent; a value stored in one rule can later be retrieved by a different rule handling a different connection. Thus, a rule can maintain persistent state across connections.

Sample Usage

```
data.set( "count", 7 );
# In another rule or connection...
$value = data.get( "count" ); # Returns 7
```

See also: data.set, data.remove, data.getMemoryUsage

5.2.30 data.getMemoryFree()

Returns the amount of free space, in bytes, available for storing information with data.set().

If memory space is low, then data.reset() can be used to clear some or all of the entries from the storage. Alternatively, the upper limit on memory can be configured using trafficscript!data_size on the Global Settings page.

Sample Usage

```
# If running low on storage, clear some temporary
# data that other rules have stored.
$bytes = data.getMemoryFree();
if( $bytes < 1024 ) {
   data.reset( "temp-" );
}
```

See also: data.set, data.reset, data.getMemoryUsage

5.2.31 data.getMemoryUsage()

Returns an estimate of the amount of memory, in bytes, used by entries that have been stored by data.set().

This can be used to verify if a rule is storing excessive amounts of data, starving the host machine of memory.

Sample Usage

```
$bytes = data.getMemoryUsage();
```

See also: data.set, data.reset, data.getMemoryFree

5.2.32 data.remove(key)

Removes the value that was previously associated with the given key using data.set().

data.remove() returns 1 if the item did exist, or 0 if it was not found.

Sample Usage

```
data.set( "cache-$url", $obj );
# In another rule or connection...
data.remove( "cache-$url" );
```

See also: data.set, data.get, data.reset

5.2.33 data.reset([prefix])

Removes some or all of the mappings created by data.set(). With no arguments, it removes all keys. With a single argument, it removes all keys that begin with the supplied string.

Sample Usage

```
# Free some memory if we've used too much
if( data.getMemoryUsage() > 102400 ) {
   data.reset( "mappings-" );
}
```

See also: data.set, data.remove, data.getMemoryUsage, data.getMemoryFree

5.2.34 data.set(key, value)

Stores a value, associating it with the provided key. The value can be retrieved later using data.get(), even from a different rule or connection.

The value will be stored as a string, implicit conversion of floating point numbers to strings can cause some precision loss. You can convert a floating point number into a string with no precision loss using 'string.sprintf("%f", $val)'. Arrays and hashes are serialised before storing and will be deserialised into their original form when retrieved with data.get().

To prevent memory problems, there is an upper limit on the amount of data that can be stored in the TrafficScript data storage. This means that the data.set() may fail. The upper limit can be configured using trafficscript!data_size on the Global Settings page.

data.set() returns true if the entry was stored, or false if there was no room.

Sample Usage

```
# Associate $value with $key for future retrieval
data.set( $key, $value );
# Run this code only once:
if( !data.get( "runonce" ) ) {
   # Do initialization
   # ...
   data.set( "runonce", 1 );
}
```

See also: data.get, data.remove, data.getMemoryUsage

5.2.35 event.emit(custom event name, message)

Triggers a Custom Event identified by the 'custom event name'. Actions can be associated with the Custom Event by configuring an Event Type to contain a Custom Event with the specified 'custom event name', and then associating that Event Type with an Action.

The 'custom event name' cannot contain '/' or control codes.

In addition to custom actions, a log message will be produced containing the eventid and the message.

Sample Usage

```
# Emit a debug statement.
event.emit( "debug1", "Some debug information" );
```

See also: log.info, log.warn, log.error

5.2.36 geo.getCity(ip)

Returns the city of the supplied IPv4 address, or the empty string.

Sample Usage

```
# Get this IP's city, such as Santa Clara
$city = geo.getCity( "216.250.81.96" );
```

5.2.37 geo.getCountry(ip)

Returns the country name of the supplied IPv4 address, or the empty string.

Sample Usage

```
# Get this IP's country, such as United States
$country = geo.getCountry( "216.250.81.96" );
```

5.2.38 geo.getCountryCode(ip)

Returns the two-character country code of the supplied IPv4 address, or the empty string.

Sample Usage

```
# Get this IP's country code, such as US
$countryCode = geo.getCountryCode( "216.250.81.96" );
```

5.2.39 geo.getDistanceKM(lat1, lon1, lat2, lon2)

Returns the distance in kilometres between two points on the earth's surface (identified by latitude and longitude), or -1 on error.

Sample Usage

```
# Get the distance between two lat-long points in km
$d = geo.getDistanceKM( "52.2338", "0.1529",
                        "37.4062", "-121.9765" );
```

5.2.40 geo.getDistanceMiles(lat1, lon1, lat2, lon2)

Returns the distance in miles between two points on the earth's surface (identified by latitude and longitude), or -1 on error.

Sample Usage

```
# Get the miles between two lat-long points
$d = geo.getDistanceMiles( "52.2338", "0.1529",
                           "37.4062", "-121.9765" );
```

5.2.41 geo.getIPDistanceKM(ip1, ip2)

Returns the distance in kilometres between the locations of two IPv4 addresses. It will return -1 unless both locations can be found.

Sample Usage

```
# Get the distance between two IPs in kilometres
$d = geo.getIPDistanceKM( "11.12.13.14",
                          "25.26.27.28" );
```

5.2.42 geo.getIPDistanceMiles(ip1, ip2)

Returns the distance in miles between the locations of two IPv4 addresses. It will return -1 unless both locations can be found.

Sample Usage

```
# Get the distance between two IPs in miles
$d = geo.getIPDistanceMiles( "1.2.3.4", "5.6.7.8" );
```

5.2.43 geo.getLatitude(ip)

Returns the decimal latitude of the supplied IPv4 address, or the empty string if the location is unknown. This may be accurate to city, region or only country level: to find out, check whether geo.getCity() and geo.getRegion() return empty strings.

Sample Usage

```
# Get this IP's decimal latitude (positive = north),
# such as 37.3961
$lat = geo.getLatitude( "216.250.81.96" );
```

5.2.44 geo.getLocation()

Returns a the name of the location in which this traffic manager is based.

Note that traffic managers can be assigned to locations only when using the Zeus Multi-Site Manager.

Sample Usage

```
# Run this part of the rule if the request was
# sent to a traffic manager located in Cambridge
if( geo.getLocation() == "Cambridge" ) {
   # ...
}
```

See also: geo.getLocationLonLat

5.2.45 geo.getLocationLonLat()

Returns a hash containing the longitude and latitude of the location to which this request was sent.

Note that traffic managers can be assigned to locations only when using the Zeus Multi-Site Manager and locations can be assigned a latitude and longitude only when using the Global Load Balancing feature.

Sample Usage

```
# How for away from the traffic manager did this
# request originate from?
$loc = geo.getLocationLonLat();
$ip = request.getRemoteIP();
$reqlon = geo.getLongitude( $ip );
$reqlat = geo.getLatitude( $ip );
$reqloc = geo.getCountry( $ip );
log.info( "Request was sent from " .
          $reqloc . ", " .
          geo.getDistanceMiles( $loc["latitude"],
                                $loc["longitude"],
                                $reqlat, $reqlon ) .
          " miles away from here" );
```

See also: geo.getLocation

5.2.46 geo.getLongitude(ip)

Returns the decimal longitude of the supplied IPv4 address, or the empty string if the location is unknown. This may be accurate to city, region or only country level: to find out, check whether geo.getCity() and geo.getRegion() return empty strings.

Sample Usage

```
# Get this IP's decimal longitude (positive = east),
# such as -121.962
$long = geo.getLongitude( "216.250.81.96" );
```

5.2.47 geo.getRegion(ip)

Returns the region (e.g. US state) of the supplied IPv4 address, or the empty string if the location is unknown or doesn't have a region.

Sample Usage

```
# Get this IP's region, such as California
$state = geo.getRegion( "216.250.81.96" );
```

5.2.48 geo.getRegionCode(ip)

Returns the two-character region code (e.g. US state abbreviation) of the supplied IPv4 address, or the empty string. The code for a given region may differ between software versions.

Sample Usage

```
# Get this IP's region code, such as CA
$stateCode = geo.getRegionCode( "64.69.78.223" );
```

5.2.49 http.addHeader(name, value)

Modifies the current HTTP request, adding an HTTP header with the supplied value. If the header already exists, then a duplicate header will be added to the message with the new value. The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Add a host header if it is missing
if( !http.headerExists( "Host" ) ) {
   http.addHeader( "Host", "unknown" );
}
```

See also: http.setHeader, http.getHeader, http.removeHeader, http.headerExists, http.addResponseHeader

5.2.50 http.addResponseHeader(name, value)

Adds an HTTP header to the HTTP response that will be sent back to the client. If the header already exists in the response, then this value will be appended to the existing value. The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Set a cookie to remember this user
http.addResponseHeader( "Set-Cookie",
   "id=12345678; path=/" );
```

See also: http.setResponseHeader, http.getResponseHeader, http.removeResponseHeader, http.addHeader

5.2.51 http.changeSite()

Redirect users to a new website. It is a more sophisticated version of http.redirect() that will preserve the original path that the request asked for. Note that it sends back a HTTP 301 redirect rather than the 302 response returned by http.redirect(). For instance, if the original request was for "http://www.example.com/image/image.jpg", then http.changeSite("example.co.uk") would redirect the user to "http://example.co.uk/image/image.jpg" The redirection will preserve the original path (and any query string) of the request, together with the port number and protocol. If you wish to force any of these details, then you can specify them as part of the supplied host name. e.g. http.changeSite("https://www.example.com") will always send people to a SSL-encrypted site. You can also add on a prefix to the path of the URL, e.g. http.changeSite("www.example.com/oldsite") would redirect a request for "http://www.example.com/missing/page.html" to "http://www.example.com/oldsite/missing/page.html". If the original request matches the supplied redirection, then http.changeSite() will take no action and let the request continue. This ensures that no 'redirection loops' occur.

Sample Usage

```
# Send users to our US site
if( http.getHostHeader() == "www.zeus.co.uk" ) {
   http.changeSite( "www.zeus.com" );
}
```

See also: http.sendResponse, http.redirect

5.2.52 http.cookie(name) - *deprecated*

This function has been deprecated. Use http.getCookie instead.

Returns the value of the named cookie.

5.2.53 http.doesFormParamExist(Parameter)

Checks whether a form parameter is present, either in the URL query string, or if not found and the request is a POST, from the POST body data. It returns 1 if the parameter is present, and 0 if not.

This is useful when there are form parameters with no value, e.g a query string like 'foo=bar&stuff&x=y' - the 'stuff' parameter has no value, but is present.

Sample Usage

```
# Did the user ask for fries with that?
if( http.doesFormParamExist( "fries" ) ) {
   # Do something
}
```

See also: http.getFormParam, http.listFormParamNames, http.getFormParams

5.2.54 http.getBody([count])

Returns the body data of the request. HTTP clients use the body data for sending file uploads or for HTML form parameters.

If the optional 'count' parameter is supplied, http.getBody() will only read and return this number of bytes. If count is 0, http.getBody() returns the entire request.

If the request has no body, then this returns an empty string. This function is not usable in response rules, as the body data of the request will no longer be accessible.

To read HTML form parameters, it is easier to use http.getFormParam() as that will work for GET and POST requests.

Sample Usage

```
# Read the entire request body
$body = http.getBody();
```

See also: http.getBodyLines, http.setBody, http.getResponseBody, http.getFormParams, http.listFormParamNames

5.2.55 http.getBodyLines([count])

Splits the body data of the HTTP request into individual lines and returns an array of the data.

If the optional 'count' parameter is supplied, http.getBodyLines() will only read and return this number of bytes. If count is 0, http.getBodyLines() returns the entire request.

If the request has no body, then this returns an empty array. This function is not usable in response rules, as the body data of the request will no longer be accessible.

To read HTML form parameters, it is easier to use http.getFormParams() as that will work for GET and POST requests.

Sample Usage

```
# Read the entire request body
$body = http.getBodyLines();
# If the body data is some special format we can
# understand then we can process it line-by-line
if( array.length( $body ) > 0 ) {
   if( array.shift( $body ) == "Special data" ) {
      foreach( $line in $body ) {
         # handle special body data lines
      }
   }
}
```

See also: http.getBody, http.getResponseBodyLines

5.2.56 http.getCookie(name)

Returns the named cookie in the incoming HTTP request.

http.getCookie is a helper method that makes it easier to parse the HTTP Cookie header and extract the values of that particular cookie, rather than using http.getHeader() directly.

An HTTP 'Cookie' header can have multiple values, such as

Cookie: user-id=Joe; user-type=gold

http.getCookie(...) will retrieve the 'Cookie' header line and parse it, returning the value of the cookie. If the cookie does not exist, http.getCookie() will return the empty string.

Sample Usage

```
# Get the PHP session cookie
$cookie = http.getCookie( "PHPSESSIONID" );
```

See also: http.getCookies, http.setCookie, http.removeCookie, http.getResponseCookie

5.2.57 http.getCookies()

Returns a hash containing the names of all the cookies in this request mapped to their values.

An HTTP 'Cookie' header can have multiple values, such as

Cookie: user-id=Joe; user-type=gold

http.getCookies() will retrieve the 'Cookie' header line and parse it, returning a hash mapping all the cookie names to their values.

Sample Usage

```
# List all of the cookie names and their values
$cookies = http.getCookies();
foreach( $cookie in hash.keys( $cookies ) ) {
   log.info( $cookie . ": " . $cookies[$cookie] );
}
```

See also: http.getCookie, http.setCookie, http.removeCookie, http.getResponseCookies

5.2.58 http.getFormParam(Parameter, [Separator])

Returns the %-decoded form parameter from the URL query string, or if not found and the request is a POST, from the POST body data.

If the parameter is provided twice, only the first will be returned, unless the optional separator is provided, in which case all matches will be returned, separated with this string.

Sample Usage

```
# See what drink the user wanted
$drink = http.getFormParam( "drink" );
```

See also: http.setQueryString, http.getRawURL, http.getBody, http.listFormParamNames, http.getFormParams

5.2.59 http.getFormParamNames(Separator) - *deprecated*

This function has been deprecated. Use http.listFormParamNames instead.

Returns a list containing the names of all the form parameters present in the URL query string and, if the request is a POST, in the POST body data. The names are returned as a single string, separated by the string supplied to this function. If the same parameter appears multiple times in the request, it will only appear once in the list returned by this function.

5.2.60 http.getFormParams()

Returns a hash mapping the names of all the form parameters present in the URL query string and, if the request is a POST, in the POST body data to their values. If the same parameter appears multiple times in the request then it will be mapped to an array of values in the hash that is returned by this function.

Sample Usage

```
# Log all of the form parameters and values
$params = http.getFormParams();
foreach( $param in hash.keys( $params ) ) {
   $values = $params[$param];
   if( lang.isArray( $values ) ) {
      log.info($param . "=" .
               array.join( $values, ", " )
              );
   } else {
      log.info($param . "=" . $values);
   }
}
```

See also: http.getFormParam, http.doesFormParamExist, http.listFormParamNames

5.2.61 http.getHeader(name)

Returns the value of a named HTTP header in the HTTP request, or the empty string if the header does not exist or has an empty value. The header name is automatically translated into the proper case for the lookup.

Sample Usage

```
# Get the browser name and version
$browser = http.getHeader( "User-Agent" );
# this returns the same value
$browser = http.getHeader( "user-agent" );
```

See also: http.getHeaders, http.setHeader, http.addHeader, http.removeHeader, http.headerExists, http.getResponseHeader, http.getHostHeader, http.getHeaders

5.2.62 http.getHeaderNames() - *deprecated*

This function has been deprecated. Use http.listHeaderNames instead.

Returns a list of all the headers that are present in the request.

The headers are returned as a single string, separated by spaces.

5.2.63 http.getHeaders()

Returns a hash containing all the header names in the request mapped to their values.

Sample Usage

```
# Show all the headers in the request
$headers = http.getHeaders();
foreach( $header in hash.keys( $headers ) ) {
   log.info( $header . "=" . $headers[$header] );
}
```

See also: http.getHeader, http.setHeader, http.addHeader, http.removeHeader, http.headerExists, http.getResponseHeaders, http.getHostHeader

5.2.64 http.getHostHeader()

Returns the HTTP Host header. This value is lowercased and has the port removed. Any trailing full stop is also removed. For example if the Host header is 'www.Zeus.com:80' then http.getHostHeader() returns 'www.zeus.com'.

Sample Usage

```
# Get the unambiguous Host header
$hostheader = http.getHostHeader();
```

See also: http.getHeader

5.2.65 http.getMethod()

Returns the HTTP method that was used to make the request, such as GET or POST.

Sample Usage

```
# Was this an HTTP POST?
if( http.getMethod() == "POST" ) {
   # handle POST request ...
}
```

See also: http.setMethod

5.2.66 http.getMultipartAttachment(part)

Returns the specified data out of a multipart encoded HTTP request. The data contained in the part is returned on success, or "" if it doesn't exist for that part. $1 contains the Content Type of the part, and $2 contains the complete headers for that part.

Sample Usage

```
$count = 0;
while( $data =
       http.getMultipartAttachment( $count ) ) {
  log.info( "Data was " . $data );
  log.info( "Content-Type was " . $1 );
  log.info( "Headers were " . $2 );
  $count = $count + 1;
}
```

5.2.67 http.getPath()

Returns the %-decoded path in the HTTP request URL, stripping the query string if one was provided. If there is a leading scheme and authority prefix, this is removed as well, so the URL "http://www.example.com/content?page=44" will be returned as "/content".

Sample Usage

```
# Retrieve the path
$path = http.getPath();
```

See also: http.setPath, http.getRawURL, http.getQueryString, http.getRawQueryString, http.normalizePath

5.2.68 http.getQueryString()

Returns the %-decoded query string in the URL, or the empty string if no query string was provided.

Sample Usage

```
# Was there a query string?
$qs = http.getQueryString();
if( $qs ) {
   # Handle query string ...
}
```

See also: http.setQueryString, http.getRawQueryString, http.setRawQueryString, http.getRawURL

5.2.69 http.getRawQueryString()

Returns the raw (non %-decoded) query string in the URL, or the empty string if no query string was provided.

Sample Usage

```
# Was there a query string?
$qs = http.getRawQueryString();
if( $qs ) {
   # Handle query string ...
}
```

See also: http.setRawQueryString, http.getQueryString, http.setQueryString, http.getRawURL

5.2.70 http.getRawURL()

Returns the raw (non-decoded) URL data provided by the client in the first line of the HTTP request, after the method and before the HTTP version specifier.

The raw URL data includes both the path and query string if supplied and is not decoded. It may also contain the protocol and hostname if the client sent them. It could contain %-escaped characters that can be used to disguise the contents of the URL. Use http.getPath() or http.getQueryString() to return the %-decoded version of the path or query string.

This function could return raw urls of various forms including:

- http://www.example.com/file.html
- /file.html
- /path/../file.html
- /file.html?querystring
- /file.html?qs%encoded

You can use http.normalizePath() on the path component to remove any '/../' or '/./' references.

In general it is better to use http.getPath() and http.getQueryString() to avoid the need to process the raw url.

Sample Usage

```
$rawurl = http.getRawURL();
if( string.contains( $rawurl, "%00" ) ) {
   # Something suspicious here ...
   connection.discard();
}
```

See also: http.getPath, http.getQueryString, http.getRawQueryString, string.unescape, http.normalizePath

5.2.71 http.getRequest()

Returns the full HTTP request and headers, but does not include any body data.

Sample Usage

```
# Check that the request is not too big
# for our servers
$request = http.getRequest();
if( string.len( $request ) > 2048 ) {
   http.sendResponse( "413 Request too large",
                      "text/plain",
                      "Request too large", "" );
}
```

See also: http.listHeaderNames, http.getHeader, http.getBody

5.2.72 http.getResponse()

Returns the beginning of the HTTP response (the status line and the headers), up to but not including the empty line that separates the headers from the body. The returned string does not include any body data.

Sample Usage

```
# See if the response included any
# headers that start with 'Foo'
$response = http.getResponse();
if( string.regexMatch( $response, "^Foo" )) {
   log.info( "A 'Foo' header was found" );
}
```

See also: http.listResponseHeaderNames, http.getResponseHeader, http.getResponseBody

5.2.73 http.getResponseBody([count])

Returns the body of the HTTP response.

If the response has chunked transfer encoding this function will return the de-chunked body. Similarly if the response is gzip or deflate compressed the body will be returned uncompressed.

If the optional 'count' parameter is provided, http.getResponseBody() will read and return the first 'count' bytes of the response. If count is 0, http.getResponseBody() will return the entire response.

Sample Usage

```
# Read the entire response body
$body = http.getResponseBody();
```

See also: http.setResponseBody, http.getBody

5.2.74 http.getResponseBodyLines([count])

Splits the body data of the HTTP request into individual lines and returns an array of the data.

If the response has chunked transfer encoding this function will return the de-chunked body. Similarly if the response is gzip or deflate compressed the body will be returned uncompressed.

If the optional 'count' parameter is provided, http.getResponseBodyLines() will read and return the first 'count' bytes of the response. If count is 0, http.getResponseBodyLines() will return the entire response.

Sample Usage

```
# Read the entire response body
$body = http.getResponseBodyLines();
```

See also: http.setResponseBody, http.getBodyLines

5.2.75 http.getResponseCode()

Returns the status code from the first line of the HTTP response.

Sample Usage

```
# Log 404 responses
if( http.getResponseCode() == 404 ) {
   log.info( "404 page for " . http.getPath() );
}
```

See also: http.setResponseCode

5.2.76 http.getResponseCookie(name)

Returns the value of the named cookie in the HTTP response.

http.getResponseCookie() is a helper method to make it easier to parse the HTTP Set-Cookie header and extract the values of that particular cookie, rather than using http.getResponseHeader() directly.

If the cookie does not exist, http.getResponseCookie() will return the empty string.

This function should be called in a response rule; it has no effect in a request rule.

Sample Usage

```
# Get the PHP session cookie
$cookie = http.getResponseCookie( "PHPSESSIONID" );
```

See also: http.getResponseCookies, http.setResponseCookie, http.removeResponseCookie, http.getCookie

5.2.77 http.getResponseCookies()

Returns a hash containing the names of all the cookies being set in this response, mapped to their values.

This function should be called in a response rule; it has no effect in a request rule.

Sample Usage

```
# List all of the cookies that are being set in
# the response and their values
$cookies = http.getResponseCookies();
foreach( $cookie in hash.keys( $cookies ) ) {
   log.info( $cookie . ": " . $cookies[$cookie] );
}
```

See also: http.getResponseCookie, http.setResponseCookie, http.removeResponseCookie, http.getCookies

5.2.78 http.getResponseHeader(name)

Returns the value of a named HTTP header in the HTTP response, or the empty string if the header does not exist or has an empty value. The header name is automatically translated into the proper case for the lookup.

Sample Usage

```
# Get the mime type of the response
$mime = http.getResponseHeader( "Content-Type" );
# note that the MIME type may look like
# 'text/html; charset=ISO-8859-1'
```

See also: http.setResponseHeader, http.addResponseHeader, http.removeResponseHeader, http.responseHeaderExists, http.getHeader

5.2.79 http.getResponseHeaderNames() - *deprecated*

This function has been deprecated. Use http.listResponseHeaderNames instead.

Returns a list of all the headers that are present in the response.

The headers are returned as a single string, separated by spaces.

5.2.80 http.getResponseHeaders()

Returns a hash containing all the header names in the response mapped to their values.

Sample Usage

```
# Show all the headers in the response
$headers = http.getResponseHeaders();
foreach( $header in hash.keys( $headers ) ) {
   log.info( $header . "=" . $headers[$header] );
}
```

See also: http.getResponseHeader, http.setResponseHeader, http.addResponseHeader, http.removeResponseHeader, http.responseHeaderExists, http.getHeaders

5.2.81 http.getResponseVersion()

Returns the version of the HTTP protocol being used. It returns the version string in the first line of the HTTP response, such as 'HTTP/1.1'. It will return the empty string in the case of HTTP/0.9 response.

Sample Usage

```
# Get the HTTP response version
$version = http.getResponseVersion();
```

See also: http.getVersion

5.2.82 http.getVersion()

Returns the version of the HTTP protocol being used. It returns the version string in the first line of the HTTP request, such as "HTTP/1.1". It will return the empty string in the case of HTTP/0.9, which does not have a version specifier in the request.

Sample Usage

```
# Get the HTTP version
$version = http.getVersion();
```

See also: http.getResponseVersion

5.2.83 http.headerExists(name)

Reports if a named header exists or not. It is similar to http.getHeader(), but makes it possible to distinguish between a header not being present and a header having no value.

The header name is automatically translated into the proper case for the lookup.

It returns 1 if the header exists, and 0 if it does not.

Sample Usage

```
# Add a host header if it is missing
if( !http.headerExists( "Host" ) ) {
   http.addheader( "Host", "unknown" );
}
```

See also: http.getHeader

5.2.84 http.listFormParamNames()

Returns an array containing the names of all the form parameters present in the URL query string and, if the request is a POST, in the POST body data. If the same parameter appears multiple times in the request, it will only appear once in the list returned by this function.

Sample Usage

```
# Log all of the form parameters and values
$params = http.listFormParamNames();
foreach( $param in $params ) {
   log.info($param . "=" .
            http.getFormParam($param, ","));
}
```

See also: http.getFormParam, http.doesFormParamExist

5.2.85 http.listHeaderNames()

Returns a list of all the headers that are present in the request.

The headers are returned as an array.

Sample Usage

```
# Log all of the header names and values
$headers = http.listHeaderNames();
foreach( $header in $headers ) {
   log.info($header . "=" . http.getHeader($header));
}
```

See also: http.getHeader, http.removeHeader, http.getRequest, http.listResponseHeaderNames

5.2.86 http.listResponseHeaderNames()

Returns a list of all the headers that are present in the response.

The headers are returned as an array.

Sample Usage

```
# Log all of the header names and values
$headers = http.listResponseHeaderNames();
foreach( $header in $headers ) {
   log.info($header . "=" . http.getHeader($header));
}
```

See also: http.getResponseHeader, http.removeResponseHeader, http.getResponse, http.listHeaderNames

5.2.87 http.normalizePath(url)

Flattens a decoded URL path, converting '//' to '/', '/./' to '/', and flattening '/a/../' to '/'. It returns the flattened path string.

If the file system path is invalid, this function returns the empty string. Invalid paths include those that contain disallowed characters like '\0', invalid hex-escapes, or that use '../' sequences to reference a location outside the local root.

This function should be used on the retrieved URL before attempting path matching for access control. Remember to pass in a %-decoded URL path, to prevent disguised paths and control codes from being 'hidden' in the path.

Sample Usage

```
# Check for access to /secure
$path = http.normalizePath( http.getPath() );
if( !$path ) {
   # bad path ...
} else if( string.startsWith( $path, "/secure" ) ) {
   # request to restricted area: check credentials
}
```

See also: http.getPath

5.2.88 http.redirect()

Sends back a HTTP 302 redirect response, which will send a web browser to a different URL. This is equivalent to http.sendResponse("302 Moved Temporarily", "text/html", "", "Location: " . $url);

Sample Usage

```
# Redirect all 404 error pages to our front page
if( http.getResponseCode() == 404 ) {
   http.redirect( "http://www.zeus.com/" );
}
```

See also: http.sendResponse, http.changeSite

5.2.89 http.removeCookie(name)

Removes the named cookie from the incoming HTTP request.

http.removeCookie is a helper method that makes it easier to parse the HTTP Cookie header and remove a particular cookie, rather than using http.getHeader() and http.setHeader() directly.

Sample Usage

```
# Remove the 'Priority' cookie
http.removeCookie( "Priority" );
```

See also: http.getCookie, http.setCookie, http.removeResponseCookie

5.2.90 http.removeHeader(name)

Removes a named header if it exists in the request.

Sample Usage

```
# Remove the 'Accept-Language' header if it exists
http.removeHeader( "Accept-Language" );
```

See also: http.getHeader, http.addHeader, http.setHeader, http.removeResponseHeader

5.2.91 http.removeResponseCookie(name)

Removes a cookie from the HTTP response.

This function should be called in a response rule; it has no effect in a request rule.

Sample Usage

```
# Remove the 'Priority' cookie
http.removeResponseCookie( "Priority" );
```

See also: http.getResponseCookie, http.setResponseCookie, http.removeCookie

5.2.92 http.removeResponseHeader(name)

Removes the named HTTP header from the HTTP response. The header name is automatically translated to the correct case.

Sample Usage

```
# Remove the 'Location' response header
http.removeResponseHeader( "Location" );
```

See also: http.setResponseHeader, http.getResponseHeader, http.addResponseHeader, http.scrubResponseHeaders, http.removeHeader

5.2.93 http.responseHeaderExists(name)

Reports if a named header exists in the HTTP response. It is similar to http.getResponseHeader(), but makes it possible to distinguish between a header not being present and a header having no value.

The header name is automatically translated into the proper case for the lookup.

It returns 1 if the header exists, and 0 if it does not.

Sample Usage

```
if( http.responseHeaderExists( "Location" ) ) {
   # Web server is redirecting user to
   # another location
}
```

See also: http.getResponseHeader

5.2.94 http.scrubRequestHeaders(header1, header2, ...)

Limits the allowed HTTP request headers to a known set. The allowed headers can either be passed in as a list or space separated in a single argument.

Care should be taken when using this function to ensure that the headers that are required for connection handling are let through. At the very least, the following should be allowed:

`Connection`, `Content-Length`, `Transfer-Encoding`, `Content-Type`, `Host`

For a complete list of HTTP headers, refer to RFC2616. Protocols that extend HTTP, such as WebDAV, use other headers.

Sample Usage

```
# Remove all headers, except the Connection,
# Content-Type, Transfer-Encoding, Content-Length
# and Host headers.
# These 2 examples are identical
http.scrubRequestHeaders( "Host",
   "Connection", "Content-Type",
   "Transfer-Encoding", "Content-Length" );
http.scrubRequestHeaders(
   "host connection content-type transfer-encoding".
   " content-length" );
```

See also: http.removeHeader, http.scrubResponseHeaders

5.2.95 http.scrubResponseHeaders(header1, header2, ...)

Limits the allowed HTTP response headers to a known set. The allowed headers can either be passed in as a list or space separated in a single argument.

Care should be taken when using this function to ensure that the headers that are required for connection handling are let through. At the very least, the following should be allowed:

`Connection`, `Content-Length`, `Transfer-Encoding`, `Location`

For a complete list of HTTP headers, refer to RFC2616. Protocols that extend HTTP, such as WebDAV, use other headers.

Sample Usage

```
# Remove all headers, except the Date, Connection,
# Content-Type, Transfer-Encoding, Content-Length
# and Location headers.
# These 2 examples are identical
http.scrubResponseHeaders( "Date",
   "Connection", "Content-Type",
   "Transfer-Encoding", "Content-Length",
   "Location" );
http.scrubResponseHeaders(
   "date connection content-type transfer-encoding".
   " content-length location" );
```

See also: http.removeResponseHeader, http.scrubRequestHeaders

5.2.96 http.sendResponse(code, type, body, headers)

Hands back an HTTP response to the client instead of balancing the request via a pool onto a node. It generates a correct HTTP response from the response code, content type, body data and headers supplied. Multiple headers should be separated with \r\n (note, however, that your traffic manager may override some of these headers, e.g. the 'Connection' header).

Sample Usage

```
# Deny access and close connection
http.sendResponse( "403 Permission Denied",
         "text/html", "Go away",
         "Set-Cookie: denied=Yes\r\nX-Foo: Bar");
```

See also: connection.close, connection.discard

5.2.97 http.setBody(body)

Sets the request body for this HTTP request to the supplied string, replacing any request body already present.

This also updates the 'Content-Length' header in the request to the length of the new body data.

Sample Usage

```
# Change the order!
$body = http.getBody();
$body = string.regexsub( $body, "Buy", "Sell", "g" );
http.setBody( $body );
```

See also: http.getBody, http.setResponseBody

5.2.98 http.setCookie(name, value)

Sets the value of the named cookie in the incoming HTTP request.

http.setCookie is a helper method that makes it easier to parse the HTTP Cookie header and set the value of a particular cookie, rather than using http.getHeader() and http.setHeader() directly.

An HTTP 'Cookie' header can have multiple values, such as

Cookie: user-id=Joe; user-type=gold

Sample Usage

```
# Set the priority cookie
http.setCookie( "Priority", "Gold" );
```

See also: http.getCookie, http.removeCookie, http.setResponseCookie

5.2.99 http.setHeader(name, value)

Sets the value of the named HTTP header, replacing any existing value if the header already exists.

The header name is automatically translated to the correct case before it is added.

Note that the "Connection" header is manipulated by your traffic manager and that changing its value in TrafficScript may not give the expected results.

Sample Usage

```
# Add (or replace) an X-Forwarded-By header
http.setHeader( "X-Forwarded-By",
   "Zeus ".sys.hostname() );
```

See also: http.addHeader, http.getHeader, http.headerExists, http.removeHeader, http.setResponseHeader

5.2.100 http.setIdempotent(resend)

Marks a request as resendable or non-resendable.

An *idempotent* request has no detrimental side effects, so it can safely be submitted multiple times. A simple page retrieval is generally idempotent. A non-idempotent request has a side effect - for example, it may update a database, or initiate a purchase. The HTTP/1.1 specification regards all GET, HEAD, PUT, DELETE, OPTIONS and TRACE requests as idempotent.

Your traffic manager tags these requests as 'resendable'; if the request is submitted to a back-end node and a correct response is not received, your traffic manager will resubmit the request to another back-end node. All other requests, such as POST requests are not resent if a back-end node fails to generate a correct response.

http.setIdempotent() can override this behaviour. If 'resend' has a non-zero value, this indicates that if the request is submitted to a back-end node and a correct response is not received, your traffic manager should resubmit the request to another back-end node.

If 'resend' is zero, this indicates that the request should only be attempted against one back-end node.

Note that a request cannot be resent if it has begun streaming data from the client to the node before it detects the failure. To avoid this, you can read the entire request within the TrafficScript rule, so that it is buffered in its entirety internally in the TrafficManager.

Sample Usage

```
# This request can be resent
if( http.getMethod() == "POST" ) {
   # Force the Zeus Traffic Manager
   # to read the entire HTTP body
   http.getBody();
   # Mark this request as resendable
   http.setIdempotent( 1 );
}
```

See also: request.setIdempotent

5.2.101 http.setMethod(method)

Sets the HTTP method, changing the original request.

Sample Usage

```
# Force HTTP POSTs to GETs
if( http.getMethod() == "POST" ) {
   http.setBody( "" );
   http.setMethod( "GET" );
}
```

See also: http.getMethod

5.2.102 http.setPath(url)

Replaces the path portion of the request URL with the supplied value. If the replacement value contains a '?', this function will also replace the query string; otherwise, any query string is preserved. Any control characters are %-encoded in the replacement value.

Sample Usage

```
# Make customer buy widget:
http.setPath( "/purchase?product=widget" );
```

See also: http.getPath

5.2.103 http.setQueryString(querystring)

Replaces the query-string portion of the request URL with the supplied replacement. Any control characters in the replacement are %-encoded.

Sample Usage

```
# Rewrite the query string
http.setQueryString( $newqs );
```

See also: http.getQueryString, http.setRawQueryString, http.getRawQueryString

5.2.104 http.setRawQueryString(querystring)

Replaces the query-string portion of the request URL with the supplied replacement. Unlike http.setQueryString, control characters are not encoded.

Sample Usage

```
# Rewrite the query string
http.setRawQueryString( "foo=%20bar" );
```

See also: http.getRawQueryString, http.setQueryString, http.getQueryString

5.2.105 http.setResponseBody(body, [transfer-encoding])

Sets the response body for this HTTP response to the supplied string, replacing any response body already present.

This also updates the 'Content-Length' header in the response to the length of the new body data. In addition the 'Content-Encoding' header is removed as we only ever set body data which is not encoded or compressed. If the server is still sending the original response body when this function is called, the connection to the server will be harmlessly dropped.

The optional 'transfer-encoding' parameter indicates the encoding of the body data (for example, 'chunked').

Sample Usage

```
$body = http.getResponseBody( 0 );
$body = string.regexsub( $body, "Buy", "Sell", "g" );
http.setResponseBody( $body );
```

See also: http.getResponseBody, http.setBody

5.2.106 http.setResponseCode(code, [message])

Sets the status code and message in the first line of the HTTP response.

Sample Usage

```
# Not enough credit!
http.setResponseCode( "402", "Payment required" );
```

See also: http.getResponseCode

5.2.107 http.setResponseCookie(name, value, [options])

Sets a cookie in the HTTP response. If the named cookie already exists, this function replaces its value.

The options are a semi-colon separated list of cookie options, such as "domain", "path", "expires" and "secure".

If the named cookie exists and no 'options' are provided, the current options for the named cookie are preserved.

This function may be called from a request rule or a response rule.

Sample Usage

```
# Set the priority cookie
http.setResponseCookie( "Priority", "Gold" );
# Set a username cookie, with various options
http.setResponseCookie( "Username", "mork",
   "domain=example.com; path=/cgi-bin" );
```

See also: http.getResponseCookie, http.removeResponseCookie, http.setCookie

5.2.108 http.setResponseHeader(name, value)

Sets a HTTP header in the HTTP response that will be sent back to the client. If the header already exists in the response, then it will be replaced with this new value.

Note that this function should not be used with the Connection header, i.e. setResponseHeader("Connection", value) since it may not give the expected results.

The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Change the server string
http.setResponseHeader( "Server",
   "Zeus" );
```

See also: http.addResponseHeader, http.getResponseHeader, http.removeResponseHeader, http.setHeader

5.2.109 http.cache.disable()

Prevents this response from being cached. In a request rule, this additionally prevents a cache lookup for this request.

Sample Usage

```
# don't cache static content...
if( $staticcontent ) { http.cache.disable(); }
```

See also: http.cache.enable, http.cache.setkey

5.2.110 http.cache.enable()

Performs the opposite function to http.cache.disable(), and re-enables the default caching behaviour when the dynamic caching option in the virtual server is enabled.

Sample Usage

```
# only cache what we explicitly enable
http.cache.disable(); # turn off everything
if( $agent == "googlebot" || $is_app1 ) {
   http.cache.enable();
}
```

See also: http.cache.disable, http.cache.setkey

5.2.111 http.cache.exists([poolname])

Returns 1 if the current request can (currently) be responded to from the cache, otherwise 0.

Note that even if the request is in the cache at the time of this call, it may be removed from the cache by the time that TrafficScript processing has finished and the traffic manager can send it. If a cached response must be guaranteed, http.cache.respondIfCached() should be used.

A pool name can be provided as optional argument in order to make the lookup use the specified pool. Without this argument, the lookup will use the pool previously selected with pool.select or the virtual server's default pool.

This function always returns 0 if called in a response rule.

Sample Usage

```
# Use a rate class only if the page is going to be
# served from the backend.
if( !http.cache.exists() ) {
   # The page cannot be served from the cache.
   # The traffic manager will have to get the page
   # from a back-end server, so rate-limit the
   # connection:
   rate.use( "rate" );
}
```

See also: http.cache.enable, http.cache.disable, http.cache.respondIfCached, pool.select

5.2.112 http.cache.respondIfCached([poolname])

Sends a cached response to the client without any further rule processing and without connecting to a back-end server. If no match is found in the cache or if the request does not allow cached responses, rule processing continues normally. If the response can be served from the cache, no statements after this function call will be processed and the client will get the cached page.

A pool name can be provided as optional argument in order to make the lookup use the specified pool. Without this argument, the lookup will use the pool previously selected with pool.select or the virtual server's default pool.

This function does nothing if called in a response rule.

Sample Usage

```
# Use a rate class only if the page is going to be
# served from the backend.
http.cache.respondIfCached();
#
# If we get here, the page could not be served
# from the cache. The traffic manager will have
# to get the page from a back-end server, so
# rate-limit the connection:
rate.use( "rate" );
```

See also: http.cache.enable, http.cache.disable, http.cache.exists, pool.select

5.2.113 http.cache.setkey()

Allows multiple variants of the same URL to be considered distinct objects, even if the standard 'Vary' RFC semantics would consider the pages identical. Cached objects will be stored with this key, and subsequent requests for the same URL will only match if the same key is provided. An example use is to provided different cached content based on a portion of the User-Agent field of the request. Note that successive uses of this function will overwrite the previous use rather than append the new key to it.

Sample Usage

```
# internal/external users see different pages
http.cache.setkey( $am_internal_user );
```

See also: http.cache.enable, http.cache.disable

5.2.114 http.compress.disable()

Stops this HTTP response from being compressed. This function overrides the Virtual Server Content Compression settings, so this is useful for stopping particular MIME types for certain browsers from being compressed.

Sample Usage

```
# Don't compress text/css pages
if( $contenttype == "text/css" ) {
    http.compress.disable();
}
```

See also: http.compress.enable

5.2.115 http.compress.enable()

Allows this individual HTTP response to be compressed. If this function is called for an HTTP response, then the Virtual Server settings for Content Compression are ignored, and the response will be compressed, assuming that the client supports compression. This function return 0 if it successfully enables compression

Sample Usage

```
# Compress all text pages for Gecko
if( string.startswith( $contenttype, "text/" ) &&
    string.contains( $useragent, "Gecko" ) ) {
    http.compress.enable();
}
```

See also: http.compress.disable

5.2.116 http.request.get(url, [headers], [timeout])

Issues an HTTP request for a remote web page and returns the body of the page requested. $1 is set to the HTTP response code (e.g. 200 for OK), or is 0 if there was an error. $2 is set to the Content-Type of the response (or an error message if there was an error). $3 is set to the content-headers of the response, in the format: [\r\n]*. $4 is set to the first line of the HTTP response. The entire original response may be reconstructed as $4."\r\n".$3."\r\n".$body, where $body is the result of http.request.get.

HTTPS pages can be requested by using the https:// prefix for the url.

A timeout parameter can be given (in seconds). If the request does not complete in this time, then an error will be given instead.

Requests made with http.request.get() will use keepalive connections to the destination server. Provided the request was completed successfully, these keepalive connections will be re-used by any invocation of http.request.get() or http.request.head() to the same destination host:port, from any rule, in any virtual server. If the request is to a node in a pool, the connection will be shared with other requests your traffic manager makes to the same node. If the request made with http.request.get() was unsuccessful, your traffic manager closes the connection.

Sample Usage

```
$body = http.request.get( "https://www.example.com/",
   "Cookie: foo=bar" );
if( $1 ) {
   http.sendResponse( $1, $2, $body, "" );
} else {
   log.info( "An error occurred: " . $2 );
}
```

See also: http.request.head, http.request.post

5.2.117 http.request.head(url, [headers], [timeout])

Issues an HTTP HEAD request for a remote web page. $1 is set to the HTTP response code (e.g. 200 for OK), or is 0 if there was an error. $2 is set to the Content-Type of the response (or an error message if there was an error). $3 is set to the content-headers of the response, in the format: [\r\n]*. $4 is set to the first line of the HTTP response. The entire original response may be reconstructed as $4."\r\n".$3."\r\n".$body, where $body is the result of http.request.get.

HTTPS pages can be requested by using the https:// prefix for the url.

A timeout parameter can be given (in seconds). If the request does not complete in this time, then an error will be given instead.

Requests made with http.request.head() will use keepalive connections to the destination server. Provided the request was completed successfully, these keepalive connections will be re-used by any invocation of http.request.head() or http.request.get() to the same destination host:port, from any rule, in any virtual server. If the request is to a node in a pool, the connection will be shared with other requests your traffic manager makes to the same node. If the request made with http.request.head() was unsuccessful, your traffic manager closes the connection.

Sample Usage

```
# Check that this site is working
http.request.head( "http://www.example.com/" );
if( $1 != 200 ) {
   pool.use( "Backup site" );
}
```

See also: http.request.post

5.2.118 http.request.post(url, POST data, [headers], [timeout])

Issues an HTTP POST request for a remote web page, and returns the body of the page requested. $1 is set to the HTTP response code (e.g. 200 for OK), or is 0 if there was an error. $2 is set to the Content-Type of the response (or an error message if there was an error). $3 is set to the content-headers of the response, in the format: [\r\n]*. $4 is set to the first line of the HTTP response. The entire original response may be reconstructed as $4."\r\n".$3."\r\n".$body, where $body is the result of http.request.post.

HTTPS pages can be requested by using the https:// prefix for the url.

A timeout parameter can be given (in seconds). If the request does not complete in this time, then an error will be given instead.

Requests made with http.request.post() will always create a new connection to the destination server and will *not* use an existing connection. After a request made with http.request.post() has finished successfully, its connection can be re-used by any invocation of http.request.get() or http.request.head() to the same destination host:port, from any rule, in any virtual server. If the request was to a node in a pool, the connection will also be shared with other requests your traffic manager makes to the same node. If the request made with http.request.post() was unsuccessful, your traffic manager closes the connection.

Sample Usage

```
$body = http.request.post("http://www.example.com/",
   "data",
   "Cookie: foo=bar\nContent-Type: text/plain" );
if( $1 ) {
   http.sendResponse( $1, $2, $body, "" );
}
```

See also: http.request.get, http.request.head

5.2.119 http.stream.continueFromBackend([data])

Stops streaming any data from the current rule and lets your traffic manager send remaining data from backend. The 'data' parameter can be used to send the last block to be streamed. Rule processing will finish and no further statements in this or subsequent rules will be executed. Unlike http.stream.finishResponse(), any data coming from the backend server will continue to be sent from the backend to the client normally.

Note that this function will behave exactly like http.stream.finishResponse() if run from a request rule.

Sample Usage

```
http.stream.startResponse(
  "200", "text/html", "", "Server: Zeus");
while( 1 ) {
  # read full lines but at most 4k of data:
  $data = http.stream.readBulkResponse( 4096, "\n" );
  if( string.find( $data, "foo" ) >= 0 ) {
     $data = string.replace( $data,"foo","bar" );
     # Job done, exit now. Any remaining body data
     # will continue to be sent from backend.
     http.stream.continueFromBackend( $data );
  }
  # stream data to client if not found.
  http.stream.writeResponse( $data );
}
http.stream.finishResponse();
```

See also: http.stream.startResponse, http.stream.readBulkResponse, http.stream.readResponse, http.stream.writeResponse, http.stream.finishResponse

5.2.120 http.stream.finishResponse([data])

Indicates that 'data' is the last block to be streamed for the current transaction. Rule processing will stop after http.stream.finishResponse() has been called, i.e. the remaining statements of the present rule will not be evaluated and no subsequent rules will be run.

Sample Usage

```
http.stream.startResponse(
  "200", "text/html", "", "Server: Zeus");
while( 1 ) {
  # read full lines but at most 4k of data:
  $data = http.stream.readBulkResponse( 4096, "\n" );
  if( $data == "" )
    break; # server has finished the response
  # invert server's logic:
  $data = string.replaceAllI( $data, "yes", "no" );
  http.stream.writeResponse( $data );
}
http.stream.finishResponse();
```

See also: http.stream.startResponse, http.stream.readBulkResponse, http.stream.readResponse, http.stream.writeResponse, http.sendResponse

5.2.121 http.stream.readBulkResponse(count, [delimiter])

Reads (and consumes) data from the server, so that TrafficScript can manipulate the data and send a modified version to the client.

Reads the number of bytes specified by 'count' from the body of the HTTP response supplied by the server. Data from the response is only returned up to and including the **last** occurrence of 'delimiter' if a non-empty delimiter has been specified. Unlike http.getResponseBody(), it also removes the data returned from the server's response. When the end of the response from the server has been reached, an empty string is returned.

If the delimiter partially matches at the end of the specified number of bytes, the data returned will include the full delimiter (thus returning slightly more data than specified). If the delimiter is not found in the specified number of bytes, then the specified number of bytes of data will be returned.

Sample Usage

```
# Stream a HTTP response back, changing the content
# as it is read in.
http.stream.startResponse(
  "200", "text/html", "", "Server: Zeus" );
while( 1 ) {
  # read several full lines but at most 4k of data:
  $data = http.stream.readBulkResponse( 4096, "\n" );
  if( $data == "" )
    break; # server has finished the response
  # invert server's logic:
  $data = string.replaceAllI( $data, "yes", "no" );
  http.stream.writeResponse( $data );
}
http.stream.finishResponse();
```

See also: http.stream.readResponse, http.getResponseBody, http.stream.writeResponse, http.stream.finishResponse, http.stream.startResponse

5.2.122 http.stream.readResponse(count, [delimiter])

Similar to http.stream.readBulkResponse(), but only returns response body data up to and including the **first** occurrence of delimiter. The two functions behave identically if no delimiter is provided.

Sample Usage

```
http.stream.startResponse(
  "200", "text/html", "", "Server: Zeus" );
while( 1 ) {
  # read full line but at most 4k of data:
  $data = http.stream.readResponse( 4096, "\n" );
  if( $data == "" )
    break; # server has finished the response
  # invert server's logic:
  $data = string.replaceAllI( $data, "yes", "no" );
  http.stream.writeResponse( $data );
}
http.stream.finishResponse();
```

See also: http.stream.readBulkResponse, http.getResponseBody, http.stream.writeResponse, http.stream.finishResponse, http.stream.startResponse

5.2.123 http.stream.startResponse(resp_code, content_type, [content_length, headers])

Sets up an HTTP response from which data can be streamed later by calling http.stream.writeResponse(). http.stream.startResponse() can only be called once per HTTP transaction. Only 'resp_code' and 'content_type' are mandatory arguments. However, it is recommended to specify the 'content_length' if possible. If it is provided and a valid integer, your traffic manager will not stream more than that number of bytes. A set of headers (separated by "\r\n") can be provided in the optional fourth argument. In a response rule, if no fourth argument is given, the response headers from the back-end will be sent on to the client (note, however, that your traffic manager may override some of these headers, e.g. the 'Connection' header)

Sample Usage

```
http.stream.startResponse(
  "200", "text/html", "",
  "Server: Zeus\r\nX-Hello: World");
while( 1 ) {
  # read full lines but at most 4k of data:
  $data = http.stream.readBulkResponse( 4096, "\n" );
  if( $data == "" )
    break; # server has finished the response
  # invert server's logic:
  $data = string.replaceAllI( $data, "yes", "no" );
  http.stream.writeResponse( $data );
}
http.stream.finishResponse();
```

See also: http.stream.writeResponse, http.stream.finishResponse, http.stream.readBulkResponse, http.stream.readResponse, http.sendResponse

5.2.124 http.stream.writeResponse(data)

Sends the data in the 'data' argument to the client. http.stream.writeResponse() can be called multiple times but http.stream.startResponse() must have been called beforehand.

Sample Usage

```
http.stream.startResponse(
  "200", "text/html", "", "Server: Zeus");
while( 1 ) {
  # read full lines but at most 4k of data:
  $data = http.stream.readBulkResponse( 4096, "\n" );
  if( $data == "" )
    break; # server has finished the response
  # invert server's logic:
  $data = string.replaceAllI( $data, "yes", "no" );
  http.stream.writeResponse( $data );
}
http.stream.finishResponse();
```

See also: http.stream.startResponse, http.stream.readBulkResponse, http.stream.readResponse, http.stream.finishResponse, http.sendResponse

5.2.125 java.run(Java Extension class name, [options])

Runs a named Java Extension. The Java Extension class name must be given, and extra options can also be supplied to the Extensions. (These are supplied as the 'args' attribute in the Java Extension API).

Sample Usage

```
java.run( "com.zeus.UserVerify", $user, $password );
```

5.2.126 log.error(message)

Writes an error message to the traffic managers's event log file. This log can be viewed through the UI.

Sample Usage

```
log.error( "Insert coffee to continue" );
```

See also: log.info, log.warn, event.emit

5.2.127 log.info(message)

Writes an informational message to the traffic manager's event log file. This log can be viewed through the UI.

Sample Usage

```
log.info( "Everything is OK" );
```

See also: log.warn, log.error, event.emit

5.2.128 log.warn(message)

Writes a warning message to the traffic manager's event log file. This log can be viewed through the UI.

Sample Usage

```
log.warn( "There may be trouble ahead" );
```

See also: log.info, log.error, event.emit

5.2.129 net.dns.resolveHost(hostname)

Resolves a hostname into an IPv4 address, using the DNS name servers configured on the local system. If the lookup fails, an empty string is returned.

Sample Usage

```
# Do a double-dns lookup
$rip = request.getRemoteIP();
$rhost = net.dns.resolveIP( $rip );
$ip = net.dns.resolveHost( $rhost );
if( $ip != $rip ) {
   log.warn( "Double lookup failed" );
}
```

See also: net.dns.resolveIP, net.dns.resolveHost6

5.2.130 net.dns.resolveHost6(hostname)

Resolves a hostname into an IPv6 address, using the DNS name servers configured on the local system. If the lookup fails, an empty string is returned.

Sample Usage

```
# Do a double-dns lookup for an IPv6 address
$rip = request.getRemoteIP();
$rhost = net.dns.resolveIP( $rip );
$ip = net.dns.resolveHost6( $rhost );
if( $ip != $rip ) {
   log.warn( "Double lookup failed" );
}
```

See also: net.dns.resolveHost, net.dns.resolveIP

5.2.131 net.dns.resolveIP(IP address)

Resolves an IP address to a hostname, using the DNS name servers configured on the local system.

Returns a hostname, or the IP address if the address cannot be resolved. An empty string is returned if the parameter is not a valid IP address.

Sample Usage

```
$rip = request.getRemoteIP();
$rhost = net.dns.resolveIP( $rip );
log.info( "Request from ".$rhost );
```

See also: net.dns.resolveHost, net.dns.resolveHost6

5.2.132 pool.activenodes(Pool)

Returns the number of nodes that are alive in the named pool. This will not include any nodes that have been marked as 'draining'.

Sample Usage

```
# Throw a warning if the number of live nodes
# drops to below 3
if( pool.activeNodes( "database" ) < 3 ) {
   log.warn( "Database nodes are low" );
}
# If there are less than two nodes, use a different
# pool.
# Better to use priority nodes in a pool for this
# though.
if( pool.activeNodes( "database" ) < 2 ) {
   pool.use( "database-offsite" );
} else {
   pool.use( "database" );
}
```

See also: pool.use, pool.select

5.2.133 pool.checknode(Pool, Host, Port)

Query the pool to determine the status of a node. Will return one of : "NOTINPOOL","NOSUCHPOOL","NOSUCHHOST","DEAD" "ACTIVE","DISABLED","DRAINING"

Sample Usage

```
$status = pool.checknode("FTP Server","app1",21);
if($status != "Active") {
   log.warn("FTP Server app1 unavailable ".$status);
}
```

5.2.134 pool.select(Pool, [Host, Port])

Selects a pool to load-balance this connection with. By default, the pool name should be a literal string (i.e. not dynamically generated and not containing any variables), however, if you enable the "trafficscript!variable_pool_use" global setting variables can be used too. Please refer to the Troubleshooting section of the Zeus TrafficScript Overview and Reference Manual for more information about this setting. Unlike pool.use(), your traffic manager will continue to process further request rules after this function.

If the pool named does not exist, your traffic manager will log a warning message.

Optionally, a specific machine can be specified that will be used to forward the request on to. This machine does not have to be in the pool selected, or in fact in any pool. In this mode, the selected pool is used only for its configuration settings (e.g. timeout values, SSL encryption options, etc.)

Sample Usage

```
# Use the pool named 'Content Pool'
pool.select( "Content Pool" );
# Send this request to www.zeus.com:80,
# using config from pool 'Zeus'
pool.select( "Zeus", "www.zeus.com", 80 );
```

See also: pool.use

5.2.135 pool.use(Pool, [Host, Port])

Selects a pool to load-balance this connection with, and stops processing any more rules. It must only be used in request rules.

By default the pool name should be a literal string, however, if you enable the "trafficscript!variable_pool_use" global setting, variables can be used too. Please refer to the Troubleshooting section of the Zeus TrafficScript Overview and Reference Manual for more information about this setting.

If the pool named does not exist, your traffic manager will log a warning message and use the default pool configured for the virtual server.

Optionally, a specific machine can be specified that will be used to forward the request on to. This machine does not have to be in the pool selected, or in fact in any pool. In this mode, the selected pool is used only for its configuration settings (e.g. timeout values, SSL encryption options, etc.)

Sample Usage

```
# Use the pool named 'Content Pool'
pool.use( "Content Pool" );
# Send this request to www.zeus.com:80,
# using config from pool 'Zeus'
pool.use( "Zeus", "www.zeus.com", 80 );
```

See also: pool.select

5.2.136 rate.getbacklog(class_name, [context])

Returns the number of connections that are currently waiting to be released by the supplied rate class.

Sample Usage

```
$backlog = rate.getbacklog( "gold-user",
                            request.getRemoteIP() );
if( $backlog > 10 ){
   # Tell the customer to come back later
   http.sendResponse( "503 Service Unavailable",
                      "text/html", "Go away",
                      "Retry-After: 10" );
}
```

See also: rate.use

5.2.137 rate.use(class_name, [context])

Immediately queues a connection using the named rate class.

The connection and the current TrafficScript rule is stalled until the rate class releases it, according to the rate limits defined in the class. When the connection is released, the rate.use() function returns and the TrafficScript rule continues to execute.

If rate.use() is called with the optional 'context' value, it uses a new rate class which inherits all of the rate settings from the named rate class. All connections called with the same 'context' value use the same new rate class. This allows you to shape connections based on arbitary data, such as a user name or source IP address, shaping connections from different users or source IPs independently.

If the connection has passed through the class successfully then the value 1 is returned. If the connection times out while it is queued, then the TrafficScript rule is abandoned. If the connection could not be queued because an invalid rate class name was provided, rate.use() returns 0.

Sample Usage

```
rate.use( "protect_database" );
rate.use( "limit_user",
          http.getCookie( "SessionID" ) );
```

5.2.138 rate.use.noQueue(class_name, [context])

Checks if this connection will exceed the rate limits of the named rate class. If connection is within rate limits, a value of 1 is returned and the connection is added to rate usage data. If usage has exceeded rate limits, a value of 0 is returned. If the rate class does not exist, a value of -1 will be returned.

Optionally a context value can be used to check rate limits based on a context, for example, rate limits for a specific client IP address. See rate.use for more details on context.

Unlike rate.use(), this will not queue connections if the rate limit is exceeded.

Note that calling rate.use() after rate.use.noQueue() will mean that the connection is counted twice, halving the allowed rate.

Sample Usage

```
$use = rate.use.noQueue( "protect_database" );
# usage is over rate limits.
if( $use == 0 ){
   http.sendResponse( "503 Service Unavailable",
                      "text/html", "Go away",
                      "Retry-After: 10" );
   connection.discard();
} else if( $use > 0 ){ # usage is within rate limits
   log.info( "No queueing" );
} else { # Rate class does not exist
   log.info( "Rate class doesn't exist" );
}
```

See also: rate.use

5.2.139 request.avoidNode()

Indicates that the named node should be avoided if at all possible.

When picking a node to use for a request, the traffic manager will not use any nodes that have been named by request.avoidNode() unless session persistence mandates it, or unless there are no other nodes available.

Sample Usage

```
# if we get a 503 Too Busy response, retry
if( http.getResponseCode() == 503 ) {
   if( request.getRetries() < 3 ) {
      request.avoidNode( connection.getNode() );
      request.retry();
   }
}
```

See also: request.retry, request.getRetries, connection.getNode

5.2.140 request.endsAt(offset)

Marks the end of the current request. Any more data read in from the network is not handled until the next request has started to be handled.

This function is useful to synchronise requests and responses. An example of its use would be for a line-oriented protocol such as POP3, where you wish to process each command.

It returns the entire request.

This function allows you to program layer-7 intelligence to correctly parse and manage generic TCP protocols.

Sample Usage

```
# get one line from input
$req = request.getLine();
# this is the end of the current request
request.endsAt( string.len( $req ) );
# Note: request.endsAt will return the request,
# but we've already got this in $req
```

See also: request.endsWith

5.2.141 request.endsWith(regex)

Marks the end of the current request. Any more data read in from the network is not handled until the next request has started to be handled.

This function is useful to synchronise requests and responses. An example of its use would be for a line-oriented protocol such as POP3, where you wish to process each command.

It returns the entire request.

This function allows you to program layer-7 intelligence to correctly parse and manage generic TCP protocols.

Sample Usage

```
# this is the end of the current request
$req = request.endsWith( "\n" );
```

See also: request.endsAt

5.2.142 request.get([count])

Returns the first 'count' bytes of data provided by the client in the current request. If no count parameter is provided, all data read so far is returned, which may be none unless request.get() has previously been called with a positive count. If you cannot determine how much data to read, use request.getLine or request.endsWith instead.

Warning: you can stall a connection by asking it to read more data than the remote client will provide. Combine this with request.getLength() or request.getLine() to reliably read data from a connection. For HTTP, you are required to use the HTTP specific functions like http.getBody() to read the request.

Sample Usage

```
# Get a length
$buf = request.get( 4 );
$l = string.bytesToInt( $buf );
# Now we know how much more data to ask for
$dat = request.get( 4 + $l );
```

See also: request.getLength, request.getLine, request.endsWith, request.set, response.get

5.2.143 request.getBandwidthClass()

Returns the current bandwidth class for the connection to the backend node, or an empty string if no class is set.

Sample Usage

```
$class = request.getBandwidthClass();
```

See also: request.setBandwidthClass, response.setBandwidthClass, response.getBandwidthClass

5.2.144 request.getDestIP()

Returns the original IP address that the client attempted to connect to. This will be the same as request.getLocalIP() unless the connection was redirected via firewall rules (e.g. using iptables on Linux)

Sample Usage

```
# Get the local IP address, such as "10.1.4.21" or
# "2001:200::8002:203:47ff:fea5:3085"
$ip = request.getDestIP();
```

See also: request.getDestPort, request.getRemoteIP, request.getRemotePort

5.2.145 request.getDestPort()

Returns the original network port number that the client attempted to connect to. This will be the same as request.getLocalPort() unless the connection was redirected via firewall rules (e.g. using iptables on Linux)

Sample Usage

```
# Get the port number on the traffic manager,
# such as 80
$port = request.getDestPort();
```

See also: request.getDestIP, request.getRemotePort, request.getRemoteIP

5.2.146 request.getLength()

Returns the number of bytes of data already received from the client. This can be combined with request.get() to reliably read data from a connection without stalling if no data is available.

Sample Usage

```
$data = request.get( request.getLength() );
```

See also: request.get, response.getLength

5.2.147 request.getLine([regex], [offset])

Returns a line of request data provided by the client. The line is terminated by the supplied regular expression, or by '\n'. If 'offset' is provided, request.getLine() returns the data from that offset to the terminating expression. The terminating expression is included in the returned string.

When request.getline() returns, the variable $1 is updated to point to the start of the next line in the datastream.

You can iterate through the lines of request data by using $1 as the iterator variable.

To prevent excessive data usage, if the line ending is not found within trafficscript!memory_warning bytes (configurable on the Global Settings page), then that many bytes will be returned.

Sample Usage

```
# Process the lines in the request until an empty
# line is found
$line = request.getLine( "\n" );
while( $line != "\n" ) {
   # process $line...
   $line = request.getLine( "\n", $1 );
}
```

See also: request.get, response.getLine

5.2.148 request.getLocalIP()

Returns the IP address that the client connected to, i.e. the address local to this machine.

Sample Usage

```
# Get the local IP address, such as "10.1.4.21" or
# "2001:200::8002:203:47ff:fea5:3085"
$ip = request.getLocalIP();
```

See also: request.getLocalPort, request.getDestIP, request.getRemoteIP, request.getRemotePort, response.getLocalIP, response.getLocalPort, response.getRemoteIP, response.getRemotePort

5.2.149 request.getLocalPort()

Returns the network port number that the client connected to. (e.g. port 80 is normal for a web server)

Sample Usage

```
# Get the local port, such as 80
$port = request.getLocalPort();
```

See also: request.getLocalIP, request.getRemotePort, request.getDestPort, request.getRemoteIP, response.getLocalPort, response.getLocalIP, response.getRemotePort, response.getRemoteIP

5.2.150 request.getLogEnabled(enabled)

Returns 1 if logging is enabled for this request, and 0 otherwise.

Sample Usage

```
if( request.getLogEnabled() ) {
   http.addResponseHeader( "X-Logged", "Yes" );
}
```

See also: request.setLogEnabled

5.2.151 request.getRemoteIP()

Returns the remote IP address of the client.

Sample Usage

```
# Get the remote IP address, such as "10.1.4.21"
# or "2001:200::8002:203:47ff:fea5:3085"
$ip = request.getRemoteIP();
```

See also: request.getRemotePort, request.getLocalIP, request.setRemoteIP, request.getLocalPort, response.getRemoteIP, response.getRemotePort, response.getLocalIP, response.getLocalPort

5.2.152 request.getRemotePort()

Returns the remote network port of the client's connection.

Sample Usage

```
# Get the remote port, such as 20427
$port = request.getRemotePort();
```

See also: request.getRemoteIP, request.getLocalPort, request.getLocalIP, response.getRemotePort, response.getRemoteIP, response.getLocalPort, response.getLocalIP

5.2.153 request.getRetries()

Returns the number of times that this request has been explicitly retried by request.retry().

Sample Usage

```
$code = http.getResponseCode();
if( $code == 404 || $code >= 500 ) {
  if( request.getRetries() < 3 ) {
    # Avoid the current node when we retry,
    # if possible
    request.avoidNode( connection.getNode() );
    request.retry();
  } else {
    http.sendResponse( "302 Redirect",
      "text/plain", "", "Location: /" );
  }
}
```

See also: request.retry, request.isResendable, pool.select

5.2.154 request.getToS(Type of Service)

Returns the Type of Service (ToS) of traffic going to the server. The return value is either "LOWDELAY", "THROUGHPUT", "RELIABILITY" or "NONE".

Sample Usage

```
if( request.getToS() != "LOWDELAY" ) {
   connection.sleep( 1000 );
}
```

See also: response.getToS, request.setToS

5.2.155 request.isResendable()

Test if it is possible to resend this request to a different node. It is only possible to resend a request if the entire request has been buffered up in the traffic manager, for example, by explicitly reading it in a request rule.

If the request was streamed through to the client, for example, as a large HTTP POST, it will not have been buffered and therefore cannot be resent.

Note that request.isResendable detects if it is possible to resend a request; request.setIdempotent can be used to tell the traffic manager not to automatically resend a request if it fails.

Sample Usage

```
if( request.isResendable() ) {
   log.info( "Retrying request" );
   request.retry();
}
```

See also: request.retry, request.getRetries, request.setIdempotent

5.2.156 request.retry()

Retry the request (using the currently selected pool). Load-balancing and session persistence decisions are recalculated, and the request is resubmitted - possibly to the same node as previously, although request.avoidNode() can prevent this.

If request.retry() is called, any request rules are not run again. When a new response is collected after request.retry(), the response rules are run again.

The response rule can modify the request in before resubmitting it.

It is only generally possible to resend a request if the entire request was read before the request rules completed. Otherwise, request data will have been streamed to the server and not cached. Use request.isResendable() to test for this.

request.getRetries() returns the number of times this request has already been tried.

On success, request.retry() does not return, but the response rules will be run again on the new response. On failure, request.retry() returns 0. request.retry() will do nothing if used in a request rule.

Sample Usage

```
$code = http.getResponseCode();
if( $code == 404 || $code >= 500 ) {
  if( request.getRetries() < 3 ) {
    # Avoid the current node when we retry,
    # if possible
    request.avoidNode( connection.getNode() );
    request.retry();
  } else {
    http.sendResponse( "302 Redirect",
      "text/plain", "", "Location: /" );
  }
}
```

See also: request.isResendable, request.getRetries, pool.select

5.2.157 request.sendResponse(Data)

Writes the provided data directly back to the client.

Any data that has been read is discarded, and nothing is forwarded to the back-end node. Once the response data has been written, control returns to the next request.

If you are managing HTTP traffic the http.sendResponse() function should be used instead.

Sample Usage

```
# Send a response
request.sendResponse( "530 Login incorrect\r\n" );
```

See also: connection.discard, http.sendResponse, response.set, response.append, connection.close

5.2.158 request.set(request data)

Replaces the input data read from the client with the provided string.

This is a low-level routine that should be used with care. For protocols with their own higher-level protocol managers (e.g. HTTP), use the higher level routines to alter the input data (e.g. http.setHeader() and http.setBody()).

Sample Usage

```
$data = request.get();
$data = string.regexsub( $data, "From", "To", "g" );
request.set( $data );
```

See also: request.get, response.set

5.2.159 request.setBandwidthClass(name)

Sets the bandwidth class for the current connection to the backend node. Providing an empty class name removes the bandwidth class from the connection. It returns zero if an error occurs (for example, if the bandwidth class does not exist), and 1 otherwise.

Sample Usage

```
request.setBandwidthClass( "gold customers" );
```

See also: request.getBandwidthClass, response.setBandwidthClass, response.getBandwidthClass

5.2.160 request.setIdempotent(resend)

Marks a request as resendable or non-resendable.

An *idempotent* request has no detrimental side effects, so it can safely be attempted multiple times. A non-idempotent request has a side effect - for example, it may update a database, or initiate a purchase.

By default, all non-HTTP requests are marked as idempotent. If a back-end node fails to generate a correct response when a request is initially forwarded to it, an attempt will be made to resend the request to another node.

request.setIdempotent() can override this behaviour. If 'resend' is zero, this indicates that the request should only be attempted against one back-end node.

If 'resend' has a non-zero value, this indicates that if a request is made to a back-end node and a correct response is not received, the request should be retried against another back-end node.

Note that a request cannot be resent once it has begun streaming data between the client and the node. Additionally, UDP connections cannot be marked as resendable (the UDP client application should handle failed UDP responses).

Sample Usage

```
# Mark this request as resendable
request.setIdempotent( 1 );
```

See also: http.setIdempotent

5.2.161 request.setLogEnabled(enabled)

Enables or disables logging for the current request. Note that if logging for the current virtual server is disabled, then this function cannot currently enable it.

Returns 1 if logging is now enabled, and 0 if it is now disabled.

Sample Usage

```
# Only log errors from the web server
if( http.getResponseCode() < 400 ) {
   request.setLogEnabled( 0 );
}
```

See also: request.getLogEnabled

5.2.162 request.setRemoteIP()

Sets the remote IP address of the client. This function should be used with care, as it will alter what is logged in request logs and the address seen by a back-end node in 'transparent' mode. 0 is returned if the IP address is invalid, and 1 otherwise.

Sample Usage

```
# Set the remote IP address, such as "10.1.4.21"
request.setRemoteIP( "10.1.4.21" );
request.setRemoteIP( "2001:200::3085" );
```

See also: request.getRemoteIP

5.2.163 request.setToS(Type of Service)

Sets the Type of Service (ToS) flags of traffic going to the server. Valid options are "LOWDELAY", "THROUGHPUT", "RELIABILITY" or "NONE". ToS flags may be used by network equipment to change how they route network traffic.

Sample Usage

```
request.setToS("LOWDELAY");
```

See also: response.setToS, request.getToS

5.2.164 request.skip([count])

Removes the specified number of bytes from the start of the request provided by the client. This can be used in combination with request.get() and request.getLine() to stream data from a client, or to alter a request before passing it on to a server.

Successive calls to this function will remove further data.

Sample Usage

```
# Skip the first 1K of data
request.skip( 1024 );
# Now skip another 1K
request.skip( 1024 );
```

See also: request.getLength, request.getLine, request.set, request.get

5.2.165 resource.exists(filename)

Checks whether or not the named file exists in *ZEUSHOME*/zxtm/conf/extra/. If it exists 1 is returned, 0 otherwise.

Sample Usage

```
# Test if the file exists
if( resource.exists( "myfile" ) ) {
   # ... process resource
}
```

See also: resource.get, resource.getMD5

5.2.166 resource.get(filename)

Returns the contents of a named file stored in the *ZEUSHOME*/zxtm/conf/extra/ directory. If the file doesn't exist, then an empty string is returned. Note that subdirectories of conf/extra are not supported.

Resources are pre-loaded into memory, so this call does not cause the file to be reloaded.

Sample Usage

```
# Read the contents of the 'info' file and add them
# as a new header.
http.addheader( "X-Info", resource.get( "info" ));
```

5.2.167 resource.getLines(filename)

Returns the contents of a named file stored in the *ZEUSHOME*/zxtm/conf/extra/ directory as an array. If the file doesn't exist, then an empty array is returned. Note that subdirectories of conf/extra are not supported.

Resources are pre-loaded into memory, so this call does not cause the file to be reloaded.

Sample Usage

```
# Read the contents of the 'info' file and add
# process them line-by-line
$info = resource.getLines( "info" );
foreach( $line in $info ) {
   # ...
}
```

5.2.168 resource.getMD5(filename)

Returns the MD5 of the current contents of the file in *ZEUSHOME*/zxtm/conf/extra/. If the file doesn't exist, an empty string is returned.

File MD5s are cached to speed up this call.

Sample Usage

```
# Get the MD5 of the file
$md5 = resource.getMD5( "myfile" );
```

See also: resource.exists, resource.get

5.2.169 resource.getMTime(filename)

Returns the time that the named file in *ZEUSHOME*/zxtm/conf/extra/ was last modified, in seconds since the epoch (i.e. UNIX time). If the file doesn't exist, 0 is returned.

Sample Usage

```
# Find out the time the file was last modified
$mtime = resource.getMTime( "myfile" );
```

See also: resource.exists, resource.get, resource.getMD5, sys.timeToString, sys.time

5.2.170 response.append(response data)

Appends the provided string to the response data.

This is a low-level routine that should be used with care. For protocols with their own higher-level protocol managers (e.g. HTTP), you must use the higher level routines to alter the input data (e.g. http.setResponseHeader() and http.setResponseBody()).

Sample Usage

```
response.append(
   "I always have to have the last word." );
```

See also: response.set

5.2.171 response.close()

Immediately closes the connection to the back-end node. Any response data that has already been read from the server will be forwarded to the client, but no more response data will be read.

Your traffic manager will reconnect to a back-end node when it next needs to forward request data to it; the back-end node is specified by either calling pool.use() or pool.select() in a request rule, or by the default pool.

Sample Usage

```
if( $neednewnode ) {
   response.close();
   pool.use( "servers" );
}
```

See also: connection.close, connection.discard, pool.use, pool.select

5.2.172 response.flush(count)

Transfers the first count bytes of the response back to the client. These bytes are removed from the underlying response buffer. If count is not specified, all current response data is flushed.

This function is useful in generic client- and server-first protocols, to synchronise responses with the next request. This may be necessary if your traffic manager is likely to respond directly to some requests, and the back-end node responds to others.

Sample Usage

```
# keep flushing response data until we get
# an empty line...
$res = response.getLine();
while( $res != "\n" ) {
   response.flush( string.len( $res ) );
   $res = response.getLine();
}
# the remainder of the response buffer will be
# flushed when all response rules complete
```

See also: response.get, response.getLine

5.2.173 response.get([count])

Returns the first 'count' bytes of data provided by the server in the current response. If you do not supply a count parameter, then the entire response will be read in.

Warning: you can stall a connection by asking it to read more data than the back-end server will provide. Combine this with response.getLength() or response.getLine() to reliably read data from a connection. For HTTP, you must use the HTTP specific functions like http.getResponseBody() to read the response.

Sample Usage

```
# Get the first 1K of data
$data = response.get( 1024 );
```

See also: response.getLength, response.getLine, response.set, request.get

5.2.174 response.getBandwidthClass()

Returns the current bandwidth class for the connection to the client, or an empty string if no class is set.

Sample Usage

```
$class = response.getBandwidthClass();
```

See also: response.setBandwidthClass, request.setBandwidthClass, request.getBandwidthClass

5.2.175 response.getLength()

Returns the amount of data already received from the server. This can be combined with response.get() to reliably read data from a connection without stalling if no data is available.

Sample Usage

```
$data = response.get( response.getLength() );
```

See also: response.get, request.getLength

5.2.176 response.getLine([regex], [offset])

Returns a line of response data provided by the server. The line is terminated by the supplied regular expression, or by '\n'. If 'offset' is provided, response.getLine() returns the data from that offset to the terminating expression.

When response.getLine() returns, the variable $1 is updated to point to the start of the next line in the datastream.

You can iterate through the lines of response data by using $1 as the iterator variable.

Sample Usage

```
# Process the lines in the response until an empty
 # line is found
 $line = response.getLine( "\n" );
 while( $line != "\n" ) {
    # process $line...
    $line = response.getLine( "\n", $1 );
 }
```

See also: response.get, request.getLine

5.2.177 response.getLocalIP()

Returns the local IP address of the connection to the node in use, i.e. an IP address on the local machine that your traffic manager connected from. It returns the empty string if no connection exists.

Sample Usage

```
# Find the IP address we connected from, such as
# "10.1.4.21" or "2001:200::8002:203:30:40:3085"
$ip = response.getLocalIP();
```

See also: response.getLocalPort, response.getRemoteIP, response.getRemotePort, request.getLocalIP, request.getLocalPort, request.getRemoteIP, request.getRemotePort

5.2.178 response.getLocalPort()

Returns the local port of the connection to the node in use, i.e. the port number on the local machine that the traffic manager connected from. It returns 0 if there is no current connection to a node.

Sample Usage

```
$port = response.getLocalPort();
```

See also: response.getLocalIP, response.getRemotePort, response.getRemoteIP, request.getLocalPort, request.getLocalIP, request.getRemotePort, request.getRemoteIP

5.2.179 response.getRemoteIP()

Returns the remote IP address of the node used. If there is no current connection, it will return an empty string.

Sample Usage

```
# Get the IP address of the node used, such as
# "10.1.4.21" or "2001:200::8002:203:a:1:3085"
$ip = response.getRemoteIP();
```

See also: response.getRemotePort, response.getLocalIP, response.getLocalPort, request.getRemoteIP, request.getRemotePort, request.getLocalIP, request.getLocalPort

5.2.180 response.getRemotePort()

Returns the network port number on which the traffic manager connected to a node. (e.g. port 80 is normal for a web server). If there is no current connection, it will return 0.

Sample Usage

```
$port = response.getRemotePort();
```

See also: response.getRemoteIP, response.getLocalPort, response.getLocalIP, request.getRemotePort, request.getRemoteIP, request.getLocalPort, request.getLocalIP

5.2.181 response.getToS(Type of Service)

Returns the Type of Service (ToS) of traffic going to the client. The return value is either "LOWDELAY", "THROUGHPUT", "RELIABILITY" or "NONE".

Sample Usage

```
if( response.getToS() == "LOWDELAY" ) {
   connection.sleep( 1000 );
}
```

See also: request.getToS, response.setToS

5.2.182 response.set(response data)

Sets the server response to the provided string.

This is a low-level routine that should be used with care. For protocols with their own higher-level protocol managers (e.g. HTTP), you must use the higher level routines to alter the input data (e.g. http.setResponseHeader() and http.setResponseBody()).

Sample Usage

```
$data = response.get();
$data = string.regexsub( $data,
   "From: ", "To: ", "g" );
response.set( $data );
```

See also: request.sendResponse, response.append, response.get, request.set

5.2.183 response.setBandwidthClass(name)

Sets the bandwidth class for the current connection to the client. Providing an empty class name removes the bandwidth class from the connection. It returns zero if an error occurs (for example, if the bandwidth class does not exist), and 1 otherwise.

Sample Usage

```
response.setBandwidthClass( "gold customers" );
```

See also: response.getBandwidthClass, request.getBandwidthClass, request.setBandwidthClass

5.2.184 response.setToS(Type of Service)

Sets the Type of Service (ToS) flags of traffic going to the client. Valid options are "LOWDELAY", "THROUGHPUT", "RELIABILITY" or "NONE". ToS flags may be used by network equipment to change how they route network traffic.

Sample Usage

```
response.setToS("LOWDELAY");
```

See also: request.setToS, response.getToS

5.2.185 rtsp.addRequestHeader(name, value)

Adds an RTSP header. If the header already exists, then this value will be appended to the existing value. The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Add a transport header
$my_header = "RTP/AVP/TCP; interleaved=0-1";
rtsp.addRequestHeader( "Transport", $my_header );
```

See also: rtsp.setRequestHeader, rtsp.getRequestHeader, rtsp.removeRequestHeader

5.2.186 rtsp.addResponseHeader(name, value)

Adds an RTSP header to the RTSP response that will be sent back to the client. If the header already exists in the response, then this value will be appended to the existing value. The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Add a new transport header
$my_header = "RTP/AVP/TCP; interleaved=0-1";
rtsp.addResponseHeader( "Transport", $my_header );
```

See also: rtsp.setResponseHeader, rtsp.getResponseHeader, rtsp.removeResponseHeader, rtsp.addRequestHeader

5.2.187 rtsp.getMethod()

Returns the RTSP method that was used to make the request, such as SETUP or PLAY.

Sample Usage

```
# Direct DESCRIBE requests to separate pool
if( rtsp.getMethod() == "DESCRIBE" ) {
   # set pool to Describe pool
}
```

See also: rtsp.setMethod

5.2.188 rtsp.getPath()

Returns the %-decoded path in the RTSP request URL

Sample Usage

```
# Retrieve the requested file
$file = rtsp.getPath();
```

See also: rtsp.setPath, rtsp.getRawURL

5.2.189 rtsp.getRawURL()

Returns the raw URL data provided by the client in the first line of the RTSP request, after the method and before the RTSP version specifier. No %-decoding is performed on the URL.

Sample Usage

```
$rawurl = rtsp.getRawURL();
if( string.contains( $rawurl, "../" ) ) {
   # Something suspicious here ...
   connection.discard();
}
```

See also: rtsp.getPath

5.2.190 rtsp.getRequest()

Returns the full RTSP request and headers, but does not include any body data.

Sample Usage

```
# Get the complete rtsp request
$request = rtsp.getRequest();
```

See also: rtsp.listRequestHeaderNames, rtsp.getRequestHeader, rtsp.getRequestBody

5.2.191 rtsp.getRequestBody([count])

Returns the body data of the request.

If the optional 'count' parameter is supplied, rtsp.getRequestBody() will only read and return this number of bytes. If count is 0, rtsp.getRequestBody() returns the entire request.

If the request has no body, then this returns an empty string. This function is not useable in response rules, as the body data of the request will no longer be accessible.

Sample Usage

```
# Read the entire request body
$body = rtsp.getRequestBody();
```

See also: rtsp.getResponseBody, rtsp.setRequestBody, rtsp.getRequestBodyLines

5.2.192 rtsp.getRequestBodyLines(count)

Splits the body data of the RTSP request into individual lines and returns an array of the data.

If the request has no body, then this returns an empty array.

Sample Usage

```
# Read the entire request body
$body = rtsp.getRequestBodyLines();
# Process it line-by-line
foreach( $line in $body ) {
   # ...
}
```

See also: rtsp.getRequestBody, rtsp.getResponseBodyLines, rtsp.setRequestBody

5.2.193 rtsp.getRequestHeader(name)

Returns the value of a named RTSP header in the RTSP request, or the empty string if the header does not exist. The header name is automatically translated into the proper case for the lookup.

Sample Usage

```
# Get the transport detail
$transport = rtsp.getRequestHeader( "Transport" );
```

See also: rtsp.setRequestHeader, rtsp.addRequestHeader, rtsp.removeRequestHeader, rtsp.getRequestHeaders

5.2.194 rtsp.getRequestHeaderNames() - *deprecated*

This function has been deprecated. Use rtsp.listRequestHeaderNames instead.

Returns a list of all the headers that are present in the request.

The headers are returned as a single string, separated by spaces.

5.2.195 rtsp.getRequestHeaders()

Returns a hash containing all the header names in the request mapped to their values.

Sample Usage

```
# Show all the headers in the request
$headers = rtsp.getRequestHeaders();
foreach( $header in hash.keys( $headers ) ) {
   log.info( $header . "=" . $headers[$header] );
}
```

See also: rtsp.getRequestHeader, rtsp.setRequestHeader, rtsp.addRequestHeader, rtsp.removeRequestHeader, rtsp.requestHeaderExists, rtsp.getResponseHeaders

5.2.196 rtsp.getResponse()

Returns the full RTSP response and headers, but does not include any body data.

Sample Usage

```
# Get the complete rtsp response
$request = rtsp.getResponse();
```

See also: rtsp.listResponseHeaderNames, rtsp.getResponseHeader, rtsp.getResponseBody

5.2.197 rtsp.getResponseBody([count])

Returns the body of the RTSP response. This could be an SDP response.

If the optional 'count' parameter is provided, rtsp.getResponseBody() will read and return the first 'count' bytes of the response. If count is 0, rtsp.getResponseBody() will return the entire response.

Sample Usage

```
# Read the entire response body
$body = rtsp.getResponseBody();
```

See also: rtsp.getRequestBody, rtsp.setResponseBody, rtsp.getResponseBodyLines

5.2.198 rtsp.getResponseBodyLines(count)

Splits the body data of the RTSP request into individual lines and returns an array of the data.

If the request has no body, then this returns an empty array.

Sample Usage

```
# Read the entire request body
$body = rtsp.getResponseBodyLines();
# Process it line-by-line
foreach( $line in $body ) {
   # ...
}
```

See also: rtsp.getResponseBody, rtsp.getRequestBodyLines, rtsp.setResponseBody

5.2.199 rtsp.getResponseCode()

Returns the status code from the first line of the RTSP response.

Sample Usage

```
# Is the status '200'
if( rtsp.getResponseCode() == 200 ) {
   # ...
}
```

See also: rtsp.setResponseCode

5.2.200 rtsp.getResponseHeader(name)

Returns the value of a RTSP header in the RTSP response, or the empty string if the header does not exist. The header name is automatically translated into the proper case for the lookup.

Sample Usage

```
# Get the transport header
$transport = rtsp.getResponseHeader( "Transport" );
```

See also: rtsp.setResponseHeader, rtsp.addResponseHeader, rtsp.removeResponseHeader, rtsp.responseHeaderExists, rtsp.getRequestHeader, rtsp.getResponseHeaders

5.2.201 rtsp.getResponseHeaderNames() - *deprecated*

This function has been deprecated. Use rtsp.listResponseHeaderNames instead.

Returns a list of all the headers that are present in the response.

The headers are returned as a single string, separated by spaces.

5.2.202 rtsp.getResponseHeaders()

Returns a hash containing all the header names in the response mapped to their values.

Sample Usage

```
# Show all the headers in the response
$headers = rtsp.getResponseHeaders();
foreach( $header in hash.keys( $headers ) ) {
   log.info( $header . "=" . $headers[$header] );
}
```

See also: rtsp.getResponseHeader, rtsp.setResponseHeader, rtsp.addResponseHeader, rtsp.removeResponseHeader, rtsp.responseHeaderExists, rtsp.getRequestHeaders

5.2.203 rtsp.getVersion()

Returns the version of the RTSP protocol being used. It returns the version string in the RTSP/version specifier in the first line of the RTSP request, such as 'RTSP/1.0'.

Sample Usage

```
# Get the RTSP version
$version = rtsp.getVersion();
```

5.2.204 rtsp.listRequestHeaderNames()

Returns a list of all the headers that are present in the request.

The headers are returned as a an array.

Sample Usage

```
# Log all of the header names and values
$headers = rtsp.listRequestHeaderNames();
foreach( $header in $headers ) {
   log.info($header . "=" .
            rtsp.getRequestHeader($header));
}
```

See also: rtsp.listResponseHeaderNames, rtsp.getRequestHeader, rtsp.getRequestHeaders

5.2.205 rtsp.listResponseHeaderNames()

Returns a list of all the headers that are present in the response.

The headers are returned as an array.

Sample Usage

```
# Log all of the header names and values
$headers = rtsp.listResponseHeaderNames();
foreach( $header in $headers ) {
   log.info($header . "=" .
            rtsp.getRequestHeader($header));
}
```

See also: rtsp.getResponseHeader, rtsp.removeResponseHeader, rtsp.getResponseHeaders, rtsp.listRequestHeaderNames

5.2.206 rtsp.redirect(path)

Sends back an RTSP 302 redirect response, which will send the client to a different URL. This is equivalent to rtsp.sendResponse("302 Moved Temporarily", "", "Location: " . $url);

Sample Usage

```
# Redirect requests for a particular file elsewhere
$path = rtsp.getPath();
if( $path == "specialfile" ) {
   rtsp.redirect( "rtsp://otherserver/" . $path );
}
```

See also: rtsp.sendResponse

5.2.207 rtsp.removeRequestHeader(name)

Removes an RTSP header if it exists in the request. The header name is automatically translated to the correct case.

Sample Usage

```
# Remove the transport header
rtsp.removeRequestHeader( "Transport" );
```

See also: rtsp.setRequestHeader, rtsp.getRequestHeader, rtsp.addRequestHeader

5.2.208 rtsp.removeResponseHeader(name)

Removes a RTSP header from the RTSP response. The header name is automatically translated to the correct case.

Sample Usage

```
# Remove the GET_PARAMETER header
rtsp.removeResponseHeader( "GET_PARAMETER" );
```

See also: rtsp.setResponseHeader, rtsp.getResponseHeader, rtsp.addResponseHeader, rtsp.removeRequestHeader

5.2.209 rtsp.requestHeaderExists(names)

Reports if a header exists or not. It is similar to rtsp.getRequestHeader(), but makes it possible to distinguish between a header not being present and a header having no value.

The header name is automatically translated into the proper case for the lookup.

It returns 1 if the header exists, and 0 if it does not.

Sample Usage

```
# Check for the Transport header
if( rtsp.requestHeaderExists( "Transport" ) ) {
   # Modify the transport options
}
```

5.2.210 rtsp.responseHeaderExists(name)

Reports if a named header exists in the RTSP response. It is similar to rtsp.getResponseHeader(), but makes it possible to distinguish between a header not being present and a header having no value.

The header name is automatically translated into the proper case for the lookup.

It returns 1 if the header exists, and 0 if it does not.

Sample Usage

```
# Test for the 'Transport' response header
if( rtsp.responseHeaderExists( "Transport" ) ) {
   # modify the parameters
}
```

See also: rtsp.getResponseHeader

5.2.211 rtsp.sendResponse(code, body, headers)

Sends back an RTSP response to the client instead of balancing the request via a pool onto a node. It generates a correct RTSP response from the response code, body data and headers supplied. Multiple headers should be separated with \r\n.

Sample Usage

```
# Discard SET_PARAMETER requests if the server
# does not support it
if( rtsp.getMethod() == "SET_PARAMETER" ) {
   rtsp.sendResponse( "401 Unauthorised", "", "" );
}
```

5.2.212 rtsp.setMethod(method)

Sets the RTSP method to use when forwarding the request via a pool to a node.

Sample Usage

```
# Force the client to send an OPTIONS packet
rtsp.setMethod( "OPTIONS" );
```

See also: rtsp.getMethod

5.2.213 rtsp.setPath(url)

Replaces the Path portion of the request URL with the supplied value.

Sample Usage

```
# Make customer view specific file:
rtsp.setPath( "myvideo.rm" );
```

See also: rtsp.getPath

5.2.214 rtsp.setRequestBody(body)

Sets the request body for this RTSP request, replacing any request body already present.

This also updates the 'Content-Length' header in the request to the length of the new body data.

Sample Usage

```
$body = rtsp.getRequestBody( 0 );
$body = string.regexsub( $body, "Buy", "Sell", "g" );
rtsp.setRequestBody( $body );
```

See also: rtsp.getRequestBody, rtsp.setResponseBody

5.2.215 rtsp.setRequestHeader(name, value)

Sets the value of a RTSP header, replacing any existing value if the header already exists.

Note that this function should not be used with the Connection header, i.e. setRequestHeader("Connection", value) since it may not give the expected results.

The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Replace the transport header
$my_header = "RTP/AVP/TCP; interleaved=0-1";
rtsp.setRequestHeader( "Transport", $my_header );
```

See also: rtsp.getRequestHeader, rtsp.addRequestHeader, rtsp.removeRequestHeader

5.2.216 rtsp.setResponseBody(body)

Sets the response body for this RTSP response, replacing any response body already present.

This also updates the 'Content-Length' header in the response to the length of the new body data. If the server is still sending the original response body when this function is called, the connection to the server will be harmlessly dropped.

Sample Usage

```
$body = rtsp.getResponseBody( 0 );
$body = string.regexsub( $body, "Buy", "Sell", "g" );
rtsp.setResponseBody( $body );
```

See also: rtsp.getResponseBody, rtsp.setRequestBody

5.2.217 rtsp.setResponseCode(code, [message])

Sets the status code and message in the first line of the RTSP response.

Sample Usage

```
# Stop clients receiving a particular file
$path = rtsp.getPath();
if( $path == "specialfile" ) {
   rtsp.setResponseCode( "401", "Unauthorised" );
}
```

See also: rtsp.getResponseCode

5.2.218 rtsp.setResponseHeader(name, value)

Sets a RTSP header in the RTSP response that will be sent back to the client. If the header already exists in the response, then it will be replaced with this new value.

Note that this function should not be used with the Connection header, i.e. setResponseHeader("Connection", value) since it may not give the expected results.

The header name is automatically translated to the correct case before it is added.

Sample Usage

```
# Replace the transport header
$my_header = "RTP/AVP/TCP; interleaved=0-1";
rtsp.setResponseHeader( "Transport", $my_header );
```

See also: rtsp.addResponseHeader, rtsp.getResponseHeader, rtsp.removeResponseHeader, rtsp.setRequestHeader

5.2.219 rule.getName()

Returns the name of the currently executing rule.

Sample Usage

```
$rulename = rule.getname();
```

5.2.220 rule.getState()

Returns the state of the currently executing rule, either "REQUEST", "RESPONSE" or "GLBRESPONSE".

Sample Usage

```
$rulestate = rule.getstate();
```

5.2.221 sip.addRequestHeader(name, value, at_top)

Modifies the current SIP request, adding a SIP header with the supplied value. If the header already exists, then this value will be appended to the existing value. If at_top is set then the value will be prepended to the header. The header name is automatically translated to the correct case before it is added.
You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Add a priority header if it is missing
if( !sip.requestHeaderExists( "Priority" ) ) {
   sip.addRequestHeader( "Priority", "normal", 0 );
}
```

See also: sip.setRequestHeader, sip.getRequestHeader, sip.removeRequestHeader, sip.requestHeaderExists

5.2.222 sip.addResponseHeader(name, value, at_top)

Adds a header to the SIP response that will be sent back to the client. If the header already exists in the response, then this value will be appended to the existing value. If at_top is set then the value will be prepended to the existing value. The header name is automatically translated to the correct case before it is added. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Use an internal webpage to see if the callee is
# logged into their system
$loggedin = http.request.get(
                  "http://internal.example.com/".
                  "lookupuser=bob" );
if( $loggedin = "No" ) {
   # Set a warning if they are not
   sip.addResponseHeader( "Warning",
      "399 zeus.com User is not logged in. ".
      " Your call might not be answered.", 0 );
}
```

See also: sip.setResponseHeader, sip.getResponseHeader, sip.removeResponseHeader, sip.addRequestHeader

5.2.223 sip.getMethod()

Returns the SIP method that was used to make the request, such as INVITE or REGISTER.

Sample Usage

```
# Direct REGISTER requests to Registrar
if( sip.getMethod() == "REGISTER" ) {
   # set pool to Registrar pool
}
```

See also: sip.setMethod

5.2.224 sip.getRequest()

Returns the full SIP request and headers, but does not include any body data.

Sample Usage

```
# Get the full SIP headers
$request = sip.getRequest();
```

See also: sip.listRequestHeaderNames, sip.getRequestHeader

5.2.225 sip.getRequestBody()

Returns the data contained in the body of the request.

Sample Usage

```
# Get the inbound Session Description to check for
# unsupported content
if( sip.getMethod() == "INVITE" ) {
   $body = sip.getRequestBody();
   # inspect packet body
}
```

See also: sip.setRequestBody, sip.getResponseBody, sip.getRequestBodyLines

5.2.226 sip.getRequestBodyLines()

Splits the body data of the SIP request into individual lines and returns an array of the data.

If the request has no body, then this returns an empty array.

Sample Usage

```
# Read the request body
$body = sip.getRequestBodyLines();
# See if it's an SDP
if( string.startswith( $body[0], "v=0" ) ) {
   # Process SDP data
}
```

See also: sip.getRequestBody, sip.getResponseBodyLines, sip.setRequestBody

5.2.227 sip.getRequestHeader(name)

Returns the named SIP header in the SIP request, or the empty string if the header does not exist. The header name is automatically translated into the proper case for the lookup. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Get information about the UAC
# originating this request
$uac = sip.getRequestHeader( "User-Agent" );
# this returns the same value
$uac = sip.getRequestHeader( "user-agent" );
```

See also: sip.setRequestHeader, sip.addRequestHeader, sip.removeRequestHeader, sip.requestHeaderExists, sip.getRequestHeaders

5.2.228 sip.getRequestHeaderNames() - *deprecated*

This function has been deprecated. Use sip.listRequestHeaderNames instead.

Returns a list of all the headers that are present in the request.

The headers are returned as a single string, separated by spaces.

5.2.229 sip.getRequestHeaders()

Returns a hash containing all the header names in the request mapped to their values.

Sample Usage

```
# Show all the headers in the request
$headers = sip.getRequestHeaders();
foreach( $header in hash.keys( $headers ) ) {
   log.info( $header . "=" . $headers[$header] );
}
```

See also: sip.getRequestHeader, sip.setRequestHeader, sip.addRequestHeader, sip.removeRequestHeader, sip.requestHeaderExists, sip.getResponseHeaders

5.2.230 sip.getRequestURI()

Returns the target of the SIP request.

Sample Usage

```
# Check a status file to see if this user
# wants to accept calls.
if( sip.getRequestURI() == "sip:bob@example.com" ) {
   # see if this user is available
   $status = http.request.get(
      "http://www.example.com/status.cgi?user=bob" );
   if( $status != "available" ) {
      sip.sendResponse( "486",
                    "User is currently " . $status );
   }
}
```

See also: sip.setRequestURI

5.2.231 sip.getResponse()

Returns the full SIP response and headers, but does not include any body data.

Sample Usage

```
# Get the full SIP headers
$request = sip.getResponse();
```

See also: sip.listResponseHeaderNames, sip.getResponseHeader

5.2.232 sip.getResponseBody()

Returns the session description of the SIP response.

Sample Usage

```
# Read the entire response body
$sdp = sip.getResponseBody();
```

See also: sip.setResponseBody, sip.getRequestBody, sip.getResponseBodyLines

5.2.233 sip.getResponseBodyLines()

Splits the body data of the SIP response into individual lines and returns an array of the data.

If the response has no body, then this returns an empty array.

Sample Usage

```
# Get the response body
$sdp = sip.getResponseBodyLines();
# Process it line-by-line
foreach( $line in $sdp ) {
   # ...
}
```

See also: sip.getResponseBody, sip.setResponseBody, sip.getRequestBodyLines

5.2.234 sip.getResponseCode()

Returns the status code from the first line of the SIP response.

Sample Usage

```
# Is the status '200'
if( sip.getResponseCode() == 200 ) {
   # ...
}
```

See also: sip.setResponseCode

5.2.235 sip.getResponseHeader(name)

Returns the value of a header in the SIP response, or the empty string if the header does not exist. The header name is automatically translated into the proper case for the lookup. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Get the route all future requests will take
$rr = sip.getResponseHeader( "Record-Route" );
```

See also: sip.setResponseHeader, sip.addResponseHeader, sip.removeResponseHeader, sip.responseHeaderExists, sip.getRequestHeader, sip.getResponseHeaders

5.2.236 sip.getResponseHeaderNames() - *deprecated*

This function has been deprecated. Use sip.listResponseHeaderNames instead.

Returns a list of all the headers that are present in the response.

The headers are returned as a single string, separated by spaces.

5.2.237 sip.getResponseHeaders()

Returns a hash containing all the header names in the response mapped to their values.

Sample Usage

```
# Show all the headers in the response
$headers = sip.getResponseHeaders();
foreach( $header in hash.keys( $headers ) ) {
   log.info( $header . "=" . $headers[$header] );
}
```

See also: sip.getResponseHeader, sip.setResponseHeader, sip.addResponseHeader, sip.removeResponseHeader, sip.responseHeaderExists, sip.getRequestHeaders

5.2.238 sip.getVersion()

Returns the version of the SIP protocol being used. It returns the version string in the SIP/version specifier in the first line of the SIP request, such as 'SIP/2.0'.

Sample Usage

```
# Get the SIP version
$version = sip.getVersion();
```

5.2.239 sip.listRequestHeaderNames()

Returns a list of all the headers that are present in the request.

The headers are returned as an array.

Sample Usage

```
# Log all of the header names and values
$headers = sip.listRequestHeaderNames();
foreach( $header in $headers ) {
   log.info($header . "=" .
            sip.getRequestHeader($header));
}
```

See also: sip.getRequestHeader, sip.removeRequestHeader, sip.getRequest, sip.listResponseHeaderNames, sip.getRequestHeaders

5.2.240 sip.listResponseHeaderNames()

Returns a list of all the headers that are present in the response.

The headers are returned as an array.

Sample Usage

```
# Log all of the header names and values
$headers = sip.listResponseHeaderNames();
foreach( $header in $headers ) {
   log.info($header . "=" .
            sip.getResponseHeader($header));
}
```

See also: sip.getResponseHeader, sip.removeResponseHeader, sip.listRequestHeaderNames, sip.getResponseHeaders

5.2.241 sip.redirect(contact)

Sends back a 302 Moved Temporarily response. This response instructs the client to retry the request at the new address(es) given in the 'contact' parameter. This is equivalent to sip.sendResponse("302", "Moved Temporarily", "Contact: " . $uri, "");

Sample Usage

```
# Example's offices are closed, redirect all their
# calls to voicemail.
$user = sip.getRequestURI();
if( string.EndsWith( $user, "@example.com" ) ) {
   $username = string.left( $user,
                      string.find( $user, "@" ) );
   $contact = $username . "@voicemail.example.com";
   sip.redirect( $contact );
}
```

See also: sip.sendResponse

5.2.242 sip.removeRequestHeader(name)

Removes a header if it exists in the request. The header name is automatically translated to the correct case.
You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Filter out any custom alert tones that have
# been specified by the client
sip.removeRequestHeader( "Alert-Info" );
```

See also: sip.getRequestHeader, sip.addRequestHeader, sip.setRequestHeader

5.2.243 sip.removeResponseHeader(name)

Removes a header from the SIP response. The header name is automatically translated to the correct case.
You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Remove the server header, if it exists, to avoid
# application-specific exploits being used
sip.removeResponseHeader( "Server" );
```

See also: sip.setResponseHeader, sip.getResponseHeader, sip.addResponseHeader, sip.removeRequestHeader

5.2.244 sip.requestHeaderExists(name)

Reports if a named header exists or not. It is similar to sip.getRequestHeader(), but makes it possible to distinguish between a header not being present and a header having no value.
The header name is automatically translated into the proper case for the lookup. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.
It returns 1 if the header exists, and 0 if it does not.

Sample Usage

```
# Add a Priority header if it is missing
if( !sip.requestHeaderExists( "Priority" ) ) {
   sip.addRequestHeader( "Priority", "normal", 0 );
}
```

See also: sip.getRequestHeader

5.2.245 sip.responseHeaderExists(name)

Reports if a named header exists in the SIP response. It is similar to sip.getResponseHeader(), but makes it possible to distinguish between a header not being present and a header having no value.

The header name is automatically translated into the proper case for the lookup. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

It returns 1 if the header exists, and 0 if it does not.

Sample Usage

```
# Test for the 'Warning' response header
if( sip.responseHeaderExists( "Warning" ) ) {
   # ...
}
```

See also: sip.getResponseHeader

5.2.246 sip.sendResponse(code, reason, [headers], [body])

Sends back a SIP response to the client instead of balancing the request via a pool onto a node. The Statue-Line of the response has the form: SIP/2.0 code reason Via, Record-Route, From, To, CSeq, Call-ID and Content-Length headers are automatically added to the response. Any headers supplied in the headers parameter will also be added to the response. Multiple headers must be separated by \r\n. Any body data specified is appended to the response.

Sample Usage

```
# Send Forbidden response if the user is blacklisted
$contact = sip.getRequestHeader( "Contact" );
if( string.contains( $contact, "10.234.12.42" ) ) {
   sip.sendResponse( "403", "Forbidden" );
}
```

See also: sip.redirect

5.2.247 sip.setMethod(method)

Sets the SIP method to use when forwarding the request via a pool to a node.

Sample Usage

```
# Force non-standard PING requests to become OPTIONS
if( sip.getMethod() == "PING" ) {
   sip.setMethod( "OPTIONS" );
   sip.setRequestHeader( "Max-Forwards", "1" );
}
```

See also: sip.getMethod

5.2.248 sip.setRequestBody(body)

Sets the request body for this SIP request to the supplied string, replacing any request body already present.

This also updates the 'Content-Length' header in the request to the length of the new body data.

Sample Usage

```
$body = sip.getRequestBody();
# Forward all data from the server through an
# intermediate machine
$body = string.regexsub($body, request.getRemoteIP(),
                        "192.168.9.100", "g");
sip.setRequestBody( $body );
```

See also: sip.getRequestBody

5.2.249 sip.setRequestHeader(name, value)

Sets a SIP header, replacing any existing value if the header already exists. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Add a reference to an information page about
# a known company when a call is received from
# them, and an icon to help identify them.
if( sip.getRequestHeader( "Organization" )
                                    == "Zeus" ) {
   sip.setRequestHeader( "Call-Info",
   "<http://www.zeus.com/assets/img/logo.gif>".
                                    " ;purpose=icon,".
   "<http://www.zeus.com/about/>".
                                    " ;purpose=info" );
}
```

See also: sip.addRequestHeader, sip.getRequestHeader, sip.requestHeaderExists, sip.removeRequestHeader

5.2.250 sip.setRequestURI(uri)

Sets the target of the SIP request.

Sample Usage

```
# If the user has recently changed username, rewrite
# requests that address their old username.
if( sip.getRequestURI() == "sip:jond@example.com" ) {
   sip.setRequestURI( "sip:jdoe@example.com" );
}
```

See also: sip.getRequestURI

5.2.251 sip.setResponseBody(body)

Sets the response body for this SIP response, replacing any response body already present.

This also updates the 'Content-Length' header in the response to the length of the new body data. If the server is still sending the original response body when this function is called, the connection to the server will be harmlessly dropped.

Sample Usage

```
$body = sip.getResponseBody();
# Forward all data from the client through an
# intermediate machine
$body = string.regexsub( $body,
                         "c=.*",
                         "c=IN IP4 "
                        ."192.168.9.100",
                         "g");
sip.setResponseBody( $body );
```

See also: sip.getResponseBody, sip.setRequestBody

5.2.252 sip.setResponseCode(code, [message])

Sets the status code and message in the first line of the SIP response.

Sample Usage

```
# Redirect client to a backup server if the proxy
# has an internal error.
if( sip.getResponseCode() == "500" ) {
   sip.setResponseHeader( "Contact",
               "sip:backup.example.com" );
   sip.setResponseCode( "305", "Use Proxy" );
}
```

See also: sip.getResponseCode

5.2.253 sip.setResponseHeader(name, value)

Sets a header in the SIP response that will be sent back to the client. If the header already exists in the response, then it will be replaced with this new value.

The header name is automatically translated to the correct case before it is added. You can specify the long or short form of the header name, so 'Via' and 'v' will both match the Via header field.

Sample Usage

```
# Change the server string
sip.setResponseHeader( "Server",
        "Zeus 5.0 (Linux/i386)" );
```

See also: sip.addResponseHeader, sip.getResponseHeader, sip.removeResponseHeader, sip.setRequestHeader

5.2.254 slm.conforming([class name])

Returns the current percentage of requests that are meeting the Service Level Monitoring objectives. If no class name is provided, it will use the SLM class currently associated with this connection; if no SLM class is associated with this connection, it returns 100.

Sample Usage

```
# If the Gold customers are starting to get slow,
# gradually reroute other services...
$conforming = slm.conforming( "gold requests" );
if( ( $level == "bronze" && $conforming < 70 ) ||
    ( $level == "silver" && $conforming < 50 )) {
  # tell lower value customers to come back later to
  # reduce load on back-end node to ensure premium
  # customers get good response
  http.sendResponse( "302", "text/html", "",
     "Location: /toobusy.html" );
}
if( $conforming < 80 && $level != "gold" ) {
  # slow down rate of responding to non-premium
  # customers
  connection.sleep( 500 );
}
```

See also: slm.threshold, slm.isOK

5.2.255 slm.isOK([class_name])

Returns whether a particular Service Level Monitoring class is meeting its objectives. If no class name is provided, it will use the SLM class currently associated with this connection; if no SLM class is associated with the connection, it returns 1. This function is a convenience shorthand for 'slm.conforming() > slm.threshold()'.

Sample Usage

```
# If the search nodes are under-utilised, use this
# spare capacity to process web page requests too
if( slm.isOK( "search-nodes" )) {
  pool.use( "search+web nodes");
} else {
  pool.use( "web nodes" );
}
```

See also: slm.conforming, slm.threshold

5.2.256 slm.threshold([class_name])

returns the value of the serious_threshold setting in the given SLM class. If no class name is provided, it will use the SLM class currently associated with this connection; if no SLM class is associated with the connection, it returns 0.

Sample Usage

```
# If we are within 10% of our threshold, divert a
# portion of traffic elsewhere. If we are less than
# 25% below of our threshold, take evasive action to
# get our site's end user experience under control!
if( slm.conforming() < slm.threshold()*0.75 ) {
   # evasive action!
   log.warn(
     "Site performance requires evasive action" );
   if( $level == "bronze" ) {
      # send away low priority traffic
      http.sendResponse( "302", "text/html", "",
        "Location: /toobusy.html" );
   } else if( $level == "silver" ) {
      # slow down processing of medium traffic
      connection.sleep( 500 );
   } else if( $level == "gold" ) {
      # use reserved bandwidth QoS
      response.setBandwidthClass(
        "premium reserved bandwidth" );
  }
} else if( slm.conforming() < slm.threshold()*1.1 ) {
   # getting slow to the danger level; start
   # proactive traffic management
   if( $customer != "gold" ) {
     pool.use( "non-priority-nodes" );
  }
}
```

See also: slm.conforming, slm.isOK

5.2.257 ssl.clientCert()

Returns the PEM encoded client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate data
$cert = ssl.clientCert();
log.info( "Certificate: " . $cert );
```

See also: ssl.isSSL

5.2.258 ssl.clientCertAlgorithm()

If the connection is SSL-encrypted and the client has supplied a valid certificate, then this returns either 'rsaEncryption', 'md2withRSAEncryption', 'md5withRSAEncryption', 'sha1withRSAEncryption' or 'RSA', depending on the certificate's encryption and hash algorithms. Otherwise, it returns the empty string.

Sample Usage

```
# Display the client certificate algorithm
$alg = ssl.clientCertAlgorithm();
log.info( "Certificate alg: ".$alg );
```

See also: ssl.isSSL

5.2.259 ssl.clientCertChain()

Returns the PEM encoded client certificate chain, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate data
$cert = ssl.clientCertChain();
log.info( "Certificate Chain: " . $cert );
```

See also: ssl.isSSL, ssl.clientCert

5.2.260 ssl.clientCertEndDate()

If the connection is SSL-encrypted and the client has supplied a valid certificate, then this returns the date when the certificate is no longer valid. The date is an integer, representing seconds since the epoch.

Otherwise, this function returns 0.

Sample Usage

```
# Display the client certificate end date
$end = ssl.clientCertEndDate();
log.info( "Certificate is valid until ".
   sys.timeToString( $end ) );
```

See also: ssl.isSSL, ssl.clientCertStartDate, sys.timeToString

5.2.261 ssl.clientCertHash()

Returns a hex-encoded MD5 hash of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate hash
$hash = ssl.clientCertHash();
log.info( "Certificate hash: ".$hash );
```

See also: ssl.isSSL

5.2.262 ssl.clientCertIssuer()

Returns a string representing the issuer of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate issuer
$issuer = ssl.clientCertIssuer();
log.info( "Certificate issuer: ".$issuer );
```

See also: ssl.isSSL

5.2.263 ssl.clientCertPublicKey()

Returns a string representation of the public key of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate key
$key = ssl.clientCertPublicKey();
log.info( "Certificate key: ".$key );
```

See also: ssl.isSSL

5.2.264 ssl.clientCertSerial()

Returns the serial (in hex) of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate serial
$serial = ssl.clientCertSerial();
log.info( "Certificate serial: ".$serial );
```

See also: ssl.clientCertSerialDec

5.2.265 ssl.clientCertSerialDec()

Returns the serial (in decimal) of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate serial in decimal
$serial = ssl.clientCertSerialDec();
log.info( "Certificate serial: ".$serial );
```

See also: ssl.clientCertSerial

5.2.266 ssl.clientCertStartDate()

If the connection is SSL-encrypted and the client has supplied a valid certificate, then this returns the date when the certificate became valid. The date is an integer, representing seconds since the epoch.

Otherwise, this function returns 0.

Sample Usage

```
# Display the client certificate start date
$start = ssl.clientCertStartDate();
log.info( "Certificate is valid from ".
   sys.timeToString( $start ) );
```

See also: ssl.isSSL, ssl.clientCertEndDate, sys.timeToString

5.2.267 ssl.clientCertStatus()

Returns 'OK' if the client certificate is valid, or 'NoClientCert' if it was missing or not valid. It returns the empty string if the connection was not SSL-encrypted.

Sample Usage

```
if( ssl.clientCertStatus() != "OK" ) {
   # Handle missing client certificate ...
}
```

See also: ssl.isSSL

5.2.268 ssl.clientCertSubject()

Returns a string representing the subject of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate subject
$subject = ssl.clientCertSubject();
log.info( "Certificate subject: ".$subject );
```

See also: ssl.isSSL

5.2.269 ssl.clientCertVersion()

Returns "1", "2" or "3" denoting the version of the client certificate, or the empty string if the connection was not SSL-encrypted or if a certificate was not supplied.

Sample Usage

```
# Display the client certificate version
$version = ssl.clientCertVersion();
log.info( "Certificate version: ".$version );
```

See also: ssl.isSSL

5.2.270 ssl.clientCipher()

Returns the cipher used by the client to SSL-encrypt the connection. It returns an empty string if the connection was not SSL-encrypted.

The string returned contains the cipher algorithm, SSL version and effective cipher strength, such as:

SSL_RSA_WITH_RC4_128_SHA, version=SSLv3, bits=128

Sample Usage

```
# Get the encryption cipher
$cipher = ssl.clientCipher();
log.info( "Encrypted with ".$cipher );
```

See also: ssl.isSSL

5.2.271 ssl.clientSupportsSecureRenegotiation()

Returns true if the client is RFC 5746 compliant, else false.

Sample Usage

```
if( ssl.isSSL() ) {
   # gather statistics about client-side support
   # for RFC 5746
   if( ssl.clientSupportsSecureRenegotiation() ) {
      counter.increment( 1 );
   } else {
      counter.increment( 2 );
   }
}
```

See also: ssl.requireCert, ssl.isSSL

5.2.272 ssl.getClientCloseAlert()

Check whether your traffic manager will send the SSL client an SSL close alert prior to terminating the TCP connection. The function will return a value of 1 if close alerts are enabled, and 0 if they are disabled. The function will return -1 if the client-side connection is not established, or is not an SSL connection.

Sample Usage

```
if( ssl.isSSL() ) {
   $alert = ssl.getClientCloseAlert();
   log.info( "client close alert: " . $alert );
}
```

See also: ssl.isSSL, ssl.setClientCloseAlert, ssl.setServerCloseAlert, ssl.getServerCloseAlert

5.2.273 ssl.getServerCloseAlert()

Check whether your traffic manager will send the SSL server an SSL close alert prior to terminating the TCP connection. The function will return a value of 1 if close alerts are enabled, and 0 if they are disabled. The function will return -1 if the connection is not an SSL connection, or if the server-side connection is not yet established (i.e. it should typically only be used in response rules.)

Sample Usage

```
if( ssl.isSSL() ) {
   $alert = ssl.getServerCloseAlert();
   log.info( "server close alert: " . $alert );
}
```

See also: ssl.isSSL, ssl.setServerCloseAlert, ssl.setClientCloseAlert, ssl.getClientCloseAlert

5.2.274 ssl.getTLSServerName()

Returns the hostname provided by the client using the TLS 1.0 'server_name' extension. If this connection is not using SSL decryption or the client did use the extension then this function returns the empty string.

Sample Usage

```
$name = ssl.getTLSServerName();
log.info( "The client provided " .
          "the server name: " . $name );
```

See also: ssl.isSSL, ssl.setTLSServerName

5.2.275 ssl.isSSL()

Returns 1 if this connection from the remote client was SSL encrypted and your traffic manager has decrypted the traffic. Otherwise, it returns 0.

Sample Usage

```
if( ssl.isSSL() ) {
   # This is an SSL-encrypted connection ...
}
```

5.2.276 ssl.requireCert()

Initiates an SSL renegotiation with the client, requiring a certificate. If the client fails to provide a valid cert, the connection is closed with an SSL alert of type handshake_failure. Otherwise, rule processing continues after the re-handshake is complete. If the client had already provided a certificate, this function does nothing and rule processing continues. If the underlying connection is not SSL, the transaction is aborted (i.e., this is interpreted as a failed re-handshake). Note that using this function overrides the setting ssl!ssl3_allow_rehandshake. It is strongly recommended to check for client-side support of RFC 5746 before using it, see ssl.clientSupportsSecureRenegotiation.

Sample Usage

```
$path = http.getPath();
if( string.startsWith( $path, "/wikileaks" ) ) {
   # Only let certain people see this data ...
   if( ssl.clientSupportsSecureRenegotiation() ) {
      ssl.requireCert();
   } else {
      http.sendResponse( 200, "text/plain",
         "Please upgrade your browser", "" );
   }
}
```

See also: ssl.clientSupportsSecureRenegotiation

5.2.277 ssl.serverCert()

Returns a PEM encoded version of the entire certificate being used by your traffic manager for this connection. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$cert = ssl.serverCert();
log.info( "Server certificate: " . $cert );
```

See also: ssl.isSSL

5.2.278 ssl.serverCertAlgorithm()

Returns a description of the algorithms being used by the virtual server's current certificate. This will either be "rsaEncryption", "md2withRSAEncryption", "md5withRSAEncryption", "sha1withRSAEncryption" or "RSA". If this virtual server is not using SSL decryption (or the algorithm type is not recognised) then this function will return the empty string.

Sample Usage

```
$alg = ssl.serverCertAlgorithm();
log.info( "Server cert algorithm: " . $alg );
```

See also: ssl.isSSL

5.2.279 ssl.serverCertCommonName()

Returns the common name of the certificate being used by your traffic manager for this connection. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$common_name = ssl.serverCertCommonName();
if( http.getHostHeader() != $common_name ) {
   log.warn( "Client browser may report certificate".
      " as invalid." );
}
```

See also: ssl.isSSL

5.2.280 ssl.serverCertEndDate()

Returns the date that the certificate being used by your traffic manager for this connection became valid. The date is an integer, representing the number of seconds since the epoch. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$end = ssl.serverCertEndDate();
if( $time < sys.time() ) {
   log.warn( "Using out of date certificate!" );
}
```

See also: ssl.isSSL, ssl.serverCertStartDate, sys.time, sys.timeToString

5.2.281 ssl.serverCertHash()

Returns the hex-encoded MD5 hash of the certificate being used by your traffic manager for this connection. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$md5 = ssl.serverCertHash();
log.info( "Server Cert Hash: " . $md5 );
```

See also: ssl.isSSL

5.2.282 ssl.serverCertIssuer()

Returns a string with the issuer data of the certificate being used by your traffic manager for this connection (each field is separated by commas). If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$issuer = ssl.serverCertIssuer();
if( ssl.serverCertSubject() == $issuer ) {
   log.info( "Certificate is self-signed" );
}
```

See also: ssl.isSSL, ssl.serverCertSubject

5.2.283 ssl.serverCertName()

Returns the name of the certificate being used by the virtual server for this connection. This is the name used to identify the certificate in the UI. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$name = ssl.serverCertName();
log.info( "Using server cert: " . $name );
```

See also: ssl.isSSL

5.2.284 ssl.serverCertPublicKey()

Returns information about the public key of the certificate being used by your traffic manager for this connection. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
# Returns public key information,
# e.g. "RSA (1024 bit)"
$public_key = ssl.serverCertPublicKey();
log.info( "Certificate public key: " . $public_key );
```

See also: ssl.isSSL

5.2.285 ssl.serverCertSerial()

Returns a string with the serial number (in hex) of the certificate being used by your traffic manager for this connection. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$serial = ssl.serverCertSerial();
log.info( "Server certificate serial: " . $serial );
```

See also: ssl.isSSL

5.2.286 ssl.serverCertStartDate()

Returns the date the certificate being used by your traffic manager for this connection became valid. The date is an integer, representing the number of seconds since the epoch. If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$start = ssl.serverCertStartDate();
log.info( "Server Certificate is valid from ".
    sys.timeToString( $start ) );
```

See also: ssl.isSSL, ssl.serverCertEndDate, sys.time, sys.timeToString

5.2.287 ssl.serverCertSubject()

Returns the subject data of the certificate being used by your traffic manager for this connection (each field is separated by commas). If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
# Returns the subject information of the cert,
# e.g. C=GB, L=Cambridge, O=Zeus, OU=Dev, CN=foo.com
$subject = ssl.serverCertSubject();
log.info( "Server cert subject: " . $subject );
```

See also: ssl.isSSL, ssl.serverCertIssuer

5.2.288 ssl.serverCertVersion()

Returns the version of the certificate being used by your traffic manager for this connection (either "1", "2" or "3"). If this virtual server is not using SSL decryption then this function will return the empty string.

Sample Usage

```
$version = ssl.serverCertVersion();
log.info( "Server Cert Version: " . $version );
```

See also: ssl.isSSL

5.2.289 ssl.serverSiteName()

Returns the hostname or IP address that was used to select the current server certificate. If the default certificate was used, or the current connection is not encrypted, the empty string is returned.

Sample Usage

```
$site_name = ssl.serverSiteName();
if( ssl.isSSL() && $site_name == "" ) {
   # The default certificate was used
}
```

See also: ssl.isSSL

5.2.290 ssl.setClientCloseAlert(alertflag)

Sets whether your traffic manager will send the SSL client an SSL close alert prior to terminating the TCP connection. An value of 0 will disable close alerts, a non-zero value will enable close alerts. This setting also applies to related connections, such as data transfer channels related to FTP command channels. If the connection with the client is not an SSL connection then calling this function will do nothing.

Sample Usage

```
if( ssl.isSSL() ) {
   # Ensure close alerts are enabled.
   ssl.setClientCloseAlert( 1 );
}
```

See also: ssl.isSSL, ssl.getClientCloseAlert, ssl.setServerCloseAlert, ssl.getServerCloseAlert

5.2.291 ssl.setServerCloseAlert(alertflag)

Sets whether your traffic manager will send the SSL server an SSL close alert prior to terminating the TCP connection. A value of 0 will disable close alerts, a non-zero value will enable close alerts. This setting also applies to related connections, such as data transfer channels related to FTP command channels. Note that this function will only work on SSL connections and the server side connection must be established (i.e. it should typically only be used in response rules.)

Sample Usage

```
if( ssl.isSSL() ) {
   # Ensure close alerts are enabled.
   ssl.setServerCloseAlert( 1 );
}
```

See also: ssl.isSSL, ssl.getServerCloseAlert, ssl.setClientCloseAlert, ssl.getClientCloseAlert

5.2.292 ssl.setTLSServerName(servername)

Instructs your traffic manager to use the specified hostname when using the TLS 1.0 server_name extension. This method only works if the back end pool has SSL encryption and the server_name option enabled. Using the empty string as this function's parameter will make your traffic manager use the hostname of the node that it is connecting to.

Sample Usage

```
# Use the hostname foo in the TLS 1.0 server_name
# extension
ssl.setTLSServerName( "foo" );
```

See also: ssl.isSSL, ssl.getTLSServerName

5.2.293 ssl.sslSessionID()

Returns the session-id of the current SSL connection, or the empty string if the connection was not SSL-encrypted.

Sample Usage

```
# Get the SSL session ID
$sessionid = ssl.sslSessionID();
log.info(
   "SessionID: ".string.hexencode( $sessionid ) );
```

See also: ssl.isSSL

5.2.294 tcp.close(sock)

Close a previously opened TCP socket. A non-zero return value indicates a successful close, if 0 is returned then an error occured and an error string will be in '$1'.

If a rule fails to close a TCP socket, it will be closed automatically when the Virtual Server connection finishes.

Sample Usage

```
$sock = tcp.connect( "10.100.1.5", 7 );
tcp.close( $sock );
```

See also: tcp.connect, tcp.write, tcp.read

5.2.295 tcp.connect(ip, port, [timeout])

Create a new TCP socket to the supplied IP address and port. This function will return a socket handle that can be used by other tcp.* functions. The created TCP socket is unique to this connection, and can't be used by other connections.

An optional timeout can be specified in milliseconds. If a connection has not been established within this time the function will return 0 and $1 will be set to "timeout".

Note that if the Virtual Server connection times out before this socket has been established then the connection will be terminated.

Returns 0 on error (with an error message in $1).

Sample Usage

```
$sock = tcp.connect( "::1", 3306 );
if( ! $sock ) {
   log.error( "Error: " . $1 );
}
```

See also: tcp.write, tcp.read, tcp.close

5.2.296 tcp.read(socket, maximum, [timeout])

Read data from a previously opened TCP connection. The function waits until data is available on the TCP socket, and will then return the available data (up to the specified 'maximum'). If an error occurs an empty string will be returned and $1 will contain an error message, or 0 if the connection has been closed.

An optional timeout can be specified in milliseconds. If a no data has been read within this time the function will return an empty string and $1 will be set to "timeout".

Note that if the Virtual Server connection times out before any data has been read then the connection will terminated.

Sample Usage

```
# Read from a TCP socket, ensuring we get 1024 bytes
$amount = 1024;
$buf = '';
while( string.len( $buf ) != $amount ) {
   $data = tcp.read( $sock, $amount -
                     string.len( $buf ) );
   if( $data == "" ) {
       $buf = '';
       break;
   }
   $buf .= $data;
}
```

See also: tcp.connect, tcp.write, tcp.close

5.2.297 tcp.write(socket, data, [timeout])

Writes all of the supplied data to a TCP socket. The function will return the number of bytes written. If an error occurs then -1 will be returned and $1 will contain error message.

An optional timeout can be specified in milliseconds. After this time the number of bytes written will be returned and $1 will contain 'timeout'.

Note that if the Virtual Server connection times out before the data has been written then the connection will terminated.

Sample Usage

```
$sock = tcp.connect( "10.100.1.5", 7 );
tcp.write( $sock, "Ping\n" );
```

See also: tcp.connect, tcp.read, tcp.close

5.2.298 xml.validate(document, DTD) - *deprecated*

This function has been deprecated. Use xml.validate.dtd instead.

Validates an XML document against a DTD. It returns 1 if the document validated correctly, and 0 if it did not. It returns -1 if there was an error parsing the XML document or the DTD.

The XML processing functionality must be enabled by the software license.

5.2.299 xml.validate.dtd(document, DTD)

Validates an XML document against a DTD. It returns 1 if the document validated correctly, and 0 if it did not. It returns -1 if there was an error parsing the XML document or the DTD.

The XML processing functionality must be enabled by the software license.

Sample Usage

```
# Validate the HTTP body against the DTD stored in
# the resource 'mydtd'
$theDoc = http.getBody( 0 );
$theDTD = resource.get( "mydtd" );
if( xml.validate.dtd( $theDoc, $theDTD ) != 1 ) {
   # validation failed ...
}
```

See also: xml.validate.xsd

5.2.300 xml.validate.xsd(document, schema)

Validates an XML document against an XML schema. It returns 1 if the document validated correctly, and 0 if it did not. It returns -1 if there was an error parsing the XML document or the schema.
If the schema against which the document is being validated needs to import another schema file, it will search for it inside Catalog > Extra Files > Miscellaneous Files.
The XML processing functionality must be enabled by the software license.

Sample Usage

```
# Validate the HTTP body against the schema stored in
# the resource 'myschema'
$theDoc = http.getBody( 0 );
$theSchema = resource.get( "myschema" );
if( xml.validate.xsd( $theDoc, $theSchema ) != 1 ) {
   # validation failed ...
}
```

See also: xml.validate.dtd

5.2.301 xml.xpath.matchNodeCount(doc, nspace, query)

Applies an XPath query to an XML document (doc), using the namespaces provided in nspace. It returns the number of entries in the result node set.

It returns -1 if there was an error parsing the XML document, XML namespace or XPath query.

The XML processing functionality must be enabled by the software license.

Sample Usage

```
# How many nodes are named 'search'?
$r = xml.xpath.matchNodeCount( $theDoc, $theNS,
   "//search" );
```

See also: xml.xpath.matchNodeSet

5.2.302 xml.xpath.matchNodeSet(doc, nspace, query)

Applies an XPath query to an XML document (doc), using the namespaces provided in nspace. It returns a string representation of the result node set.

It returns the empty string if there was an error parsing the XML document, XML namespace or XPath query.

The XML processing functionality must be enabled by the software license.

Sample Usage

```
# What is the value of the 'search' node?
$s = xml.xpath.matchNodeSet( $theDoc, $theNS,
   "//search/text()" );
```

See also: xml.xpath.matchNodeCount

5.2.303 xml.xslt.transform(document, stylesheet)

Performs an XSLT transformation on a XML document. It returns the transformed document, or -1 on failure.

The XML processing functionality must be enabled by the software license.

Sample Usage

```
# Perform XSLT transformation on supplied POST XML
# data, and return the HTML result to the client.
$response = xml.xslt.transform( http.getbody( 0 ),
   resource.get( "people.xsl" ));
http.sendResponse( "200 OK", "text/html",
   $response, "" );
```

Further Resources

6.1 Zeus Manuals

Bundled with the software is a Getting Started Guide, intended to get you up and running quickly with the software, and a more detailed User Guide. If you have purchased a Zeus 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 Zeus Control API.

You can access these manuals via the **Help** pages (described below), or download the most recent versions from the Zeus KnowledgeHub at http://knowledgehub.zeus.com/ .

6.2 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.

You can access the User Guide and TrafficScript reference manual by clicking the Manuals button on any of the **Help** pages.

The **Rules > Edit** page also has a link to **TrafficScript Help**, a quick reference guide to the functions.

6.3 Information online

Product specifications can be found at:

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

Visit Zeus KnowledgeHub for further documentation, examples, white papers and other resources:

http://knowledgehub.zeus.com/

Index