Zeus Control API and Reference

Version 6.0

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 the Zeus traffic management product family

Zeus traffic management products provide high-availability, application-centric traffic management and load balancing product. They provides control, intelligence, security and resilience for all your application traffic. These products are 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.

1.2 Introducing the Control API

A cluster of traffic mangers is normally managed using the web-based Admin Server on one of the machines.

Zeus's Control API provides an alternative means to remotely administer and configure a Zeus cluster. For example, when an Intrusion Detection System detects a remote attack attempt, it could use the Control API to configure the cluster to drop all connections from the suspect IP address.

A provisioning system could detect server overloading by monitoring the response times of the server nodes using Service Level Monitoring and the SNMP interface. Once it had provisioned additional servers, it could then reconfigure the server pools using the Zeus Control API.

1.2.1 Standards-conformant SOAP communications

The Control API is a standards-conformant SOAP-based API that provides the means for other applications to query and modify the configuration of the cluster.

> *SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.*
>
> Simple Object Access Protocol (SOAP), w3.org

Most importantly, SOAP is a commonly accepted standard that allows applications to communicate. Zeus's Control API is published in the form of WSDL (Web Services Description Language) files. These files document which methods (remote procedure calls) are available, what input parameters they take and the output they return.

The WSDL files are located in `ZEUSHOME/zxtm/etc/wsdl`, and can be downloaded from the SOAP API page in the online help.

A SOAP-compliant programming environment will parse the WSDL files to determine which remote methods can be called, and will then allow the application to call these methods

much as if they were local functions. The SOAP environment insulates the application developer from the underlying complexity – network connectivity, XML formatting, cross-platform compatibility, etc. The application developer can concentrate on implementing the control logic required to support the application they are building.

The Zeus Control API can be used by any programming language and application environment that supports SOAP services. C#, Perl, Java and Python are commonly used.

1.3 SOAP-based Architecture

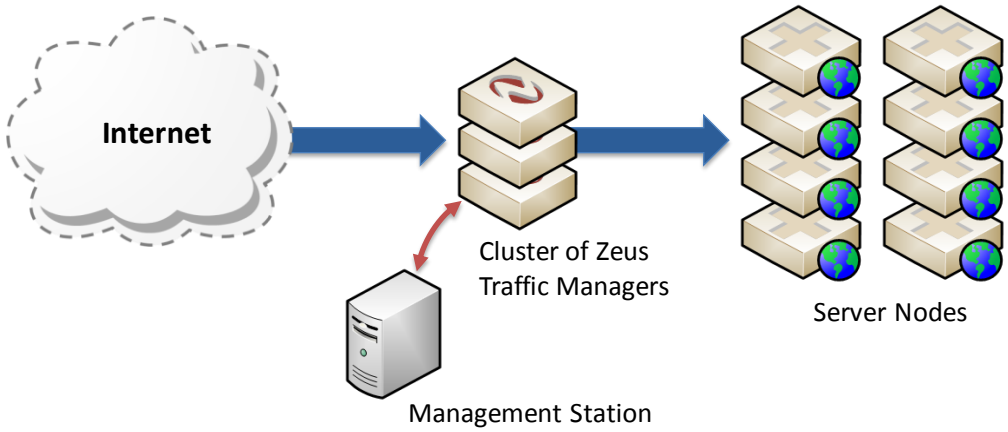

Fig. 1. Arrangement of Management Server, Zeus Cluster and Server Nodes

A management application can issue a SOAP request to one of the traffic managers in a Zeus cluster. The application may be running on a stand-alone management server, one of the server nodes, or even on one of the traffic managers.

The application can issue the request to any of the Zeus traffic managers. The traffic managers automatically synchronize their configuration, so a configuration change sent to one machine is automatically replicated across the cluster.

1.3.1 Security Considerations

The SOAP-based management application communicates with a SOAP server running on the Zeus Admin Server, so the same security considerations apply:

- If a management network or IP-based access control is in use to secure the Admin Server, these will affect the locations that the management application can run from.
- SOAP traffic is automatically encrypted using SSL.
- The Zeus Admin Server will authenticate itself with its SSL certificate, which is generally self-signed.

  You may need to ensure that your SOAP application accepts self-signed certificates, or install a trusted SSL certificate in your Zeus admin server.

- SOAP requests are authenticated using the credentials of a users who is a member of a group with 'Control API' permissions in the Administration Server. Groups are defined in the Zeus Administration Interface, in the **System** > **Users** > **Groups** page.

  By default, the 'admin' group (which includes the user named 'admin') is the only group that is permitted to use the Zeus Control API. You can add this permission to other groups as required.

  You may wish to define a specific username for your management application to use, so that you can track its activity using traffic manager's Audit Log.

Code samples

The following code samples demonstrate how to call the Zeus Control API from several different application environments. They are intended to illustrate the similarities, rather than the best practice for each language.

2.1 Listing running virutal servers

The examples connect to a traffic manager, retrieve a list of the virtual servers and then query whether each virtual server is enabled (i.e. running). They then print out the running virtual servers.

The code structure is as follows:

- Specify the location of the admin server, and the username and password of an account in the 'admin' group or another group with explicit 'Zeus Control API' permissions (see section 1.3.1).
- If necessary, configure the HTTPS layer to accept the Admin Server's self-signed certificate.
- Instantiate a means of calling the SOAP methods of the VirtualServer interface, generally with reference to the WSDL specification[1].
- Invoke the `VirtualServer:getVirtualServerNames()` method, which returns an array of string values.
- Invoke the `VirtualServer:getEnabled()` method, providing an array of string values (the names) and obtaining an array of Boolean values.
- Iterate through the arrays, printing the names of the virtual servers which are enabled.

2.1.1 listVS.pl using Perl SOAP::Lite

```
#!/usr/bin/perl -w

use SOAP::Lite 0.60;

# This is the url of the Zeus admin server
my $admin_server = 'https://username:password@host:9090';

my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap");
```

[1] Some environments, such as Perl's SOAP::Lite do not validate method calls against the WSDL specification.

```
# Get a list of virtual servers
my $res = $conn->getVirtualServerNames();
my @names = @{$res->result};

# Establish which are enabled
$res = $conn->getEnabled( \@names );
my @enabled = @{$res->result};

# Print those which are enabled
for( my $i = 0; $i <= $#names; $i++ ) {
   if( $enabled[$i] ) {
      print "$names[$i]\n";
   }
}
```

Run the example as follows:

```
$ ./listVS.pl
Main website
Mail servers
Test site
```

To run this example, you will need Perl, `SOAP::Lite` and `IO::Socket::SSL`.

- On Debian-based systems, install the packages `libsoap-lite-perl` and `libio-socket-ssl-perl`.
- On RedHat based systems, you'll need the `perl-SOAP-Lite` and `perl-IO-Socket-SSL` rpms
- Sites that use CPAN can obtain the modules from http://search.cpan.org/~byrne/SOAP-Lite-0.67/lib/OldDocs/SOAP/Lite.pm and http://search.cpan.org/~behroozi/IO-Socket-SSL-0.97/SSL.pm

Early versions of SOAP::Lite used a 'uri' method instead of the current 'ns' one. This affected versions prior to 0.65_5. If you are using a old SOAP::Lite, use the following code to create the SOAP::Lite connection instead:

```
my $conn = SOAP::Lite
    -> uri('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap");
```

Perl's `SOAP::Lite` module does not use the WSDL file to perform any type checking, so calling errors will be detected at runtime and SOAP structures and enumerations must be managed manually (see chapter 3). The SSL layer is happy to accept self-signed certificates.

2.1.2 listVS.cs using C Sharp

```
using System;
using System.Net;
using System.IO;
using System.Security.Cryptography.X509Certificates;

public class AllowSelfSignedCerts : IcertificatePolicy {
   public bool CheckValidationResult(  ServicePoint sp,
         X509Certificate cert, WebRequest request, int problem )
   {
      return true;
   }
}

public class listVS {

   public static void Main( string [] args )
   {
      System.Net.ServicePointManager.CertificatePolicy =
         new AllowSelfSignedCerts();

      string url= "https://host:9090/soap";
      string username = "username";
      string password = "password";

      try {
         Zeus.VirtualServer p = new Zeus.VirtualServer();
         p.Url = url;
         p.Credentials = new NetworkCredential( username, password );

         string[] names = p.getVirtualServerNames();
         bool[] enabled = p.getEnabled( names );

         for ( int i = 0; i < names.Length; i++  ) {
            if( enabled[i] ) {
               Console.WriteLine( "{0}", names[i] );
            }
         }
      } catch ( Exception e ) {
         Console.WriteLine( "{0}", e );
      }
   }
}
```

This code works with the .NET 1.1 SDK and with Mono[2].

Using .Net 1.1, compile and run this example as follows:

```
C:\> wsdl -o:VirtualServer.cs -n:Zeus VirtualServer.wsdl
C:\> csc /out:listVS.exe VirtualServer.cs listVS.cs
C:\> listVS.exe
Main website
Mail servers
Test site
```

With Mono, compile and run as follows:

```
$ wsdl -o:VirtualServer.cs -n:Zeus VirtualServer.wsdl
$ mcs /out:listVS.exe /r:System.Web.Services \
        VirtualServer.cs listVS.cs
$ listVS.exe
Main website
Mail servers
Test site
```

The WSDL interface specifications for the Zeus Control API are located in `ZEUSHOME/zxtm/etc/wsdl/`, and can be downloaded from the SOAP API page in the online help.

Note the use of the `IcertificatePolicy` derived class to override the default certificate checking method. This allows the application to accept the Zeus Admin Server's self-signed certificate.

[2] Use the most recent build of Mono from http://www.mono-project.com/.

2.1.3 listVS.java using Java

```
import com.zeus.soap.zxtm._1_0.*;

import java.security.Security;
import java.security.KeyStore;
import java.security.Provider;
import java.security.cert.X509Certificate;
import javax.net.ssl.ManagerFactoryParameters;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactorySpi;
import javax.net.ssl.X509TrustManager;

public class listVS {

   public static void main( String[] args ) {

      // Install the all-trusting trust manager
      Security.addProvider( new MyProvider() );
      Security.setProperty( "ssl.TrustManagerFactory.algorithm",
         "TrustAllCertificates" );

      try {
         VirtualServerLocator vsl = new VirtualServerLocator();
         vsl.setVirtualServerPortEndpointAddress(
            "https://username:password@host:9090/soap" );
         VirtualServerPort vsp = vsl.getVirtualServerPort();

         String[] vsnames = vsp.getVirtualServerNames();
         boolean[] vsenabled = vsp.getEnabled( vsnames );

         for( int i = 0; i < vsnames.length; i++ ){
            if( vsenabled[i] ){
               System.out.println( vsnames[i] );
            }
         }
      } catch (Exception e) {
         System.out.println( e.toString() );
      }

   }

   /* The following code disables certificate checking.
    * Use the Security.addProvider and Security.setProperty
```

```
     * calls to enable it */
   public static class MyProvider extends Provider {
      public MyProvider() {
         super( "MyProvider", 1.0, "Trust certificates" );
         put( "TrustManagerFactory.TrustAllCertificates",
            MyTrustManagerFactory.class.getName() );
      }

      protected static class MyTrustManagerFactory
            extends TrustManagerFactorySpi {
         public MyTrustManagerFactory() {}
         protected void engineInit( KeyStore keystore ) {}
         protected void engineInit(
            ManagerFactoryParameters mgrparams ) {}
         protected TrustManager[] engineGetTrustManagers() {
            return new TrustManager[] {
               new MyX509TrustManager()
            };
         }
      }

      protected static class MyX509TrustManager
            implements X509TrustManager {
         public void checkClientTrusted(
            X509Certificate[] chain, String authType) {}
         public void checkServerTrusted(
            X509Certificate[] chain, String authType) {}
         public X509Certificate[] getAcceptedIssuers() {
            return null;
         }
      }
   }
}
```

The bulk of this code disables client certificate checking. Details of the code and surrounding infrastructure are at:

http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html

This code works with the Java 1.5 SDK/JRE. To build and run the code, you'll need to do the following:

1. Download axis from http://ws.apache.org/axis/. This provides the WSDL-to-Java converter. Copy all the .jar files from `axis-<version>/libs/` to the *JAVAHOME*`/jre/lib/ext/` directory, or add them to your `CLASSPATH`.

2. Install the Java Activation Framework and JavaMail libraries to avoid warnings when the code is run:

   - Java Activation Framework

     http://java.sun.com/products/javabeans/glasgow/jaf.html

   - JavaMail

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

   Copy the `activation.jar` and `mail.jar` files contained in these packages to the *JAVAHOME*`/jre/lib/ext/` directory, or add them to your `CLASSPATH`.

3. In your build directory, convert the required WSDL files into java code as follows[3]:

```
$ java org.apache.axis.wsdl.WSDL2Java VirtualServer.wsdl
```

4. Compile and run the example as follows:

```
$ javac listVS.java
$ java listVS
Main website
Mail servers
Test site
```

This code uses the functions within the VirtualServer interface. Other interfaces use a similar pattern.

For example, if you wished to access functions within the `XXX` interface, you would need to instantiate an `XXXLocator` object, declare the location of the traffic manager using the `setXXXPortEndpointAddress()` function and then create a connection using `getXXXPort()` to return an `XXXPort` object. You can then invoke methods using the `XXXPort` object. Java is verbose, but generally repetitive so the patterns can be copied thus:

```
SystemCacheLocator scl = new SystemCacheLocator();
scl.setSystemCachePortEndpointAddress(
   "https://username:password@host:9090/soap" );
SystemCachePort scp = scl.getSystemCachePort();
```

[3] The WSDL interface specifications for the Zeus Control API are located in `ZEUSHOME/zxtm/etc/wsdl/`.

```
/* Invoke the methods on the SystemCachePort object */
scp.clearWebCache();
```

2.1.4 listVS.py using Python

```
#!/usr/bin/python

import SOAPpy

conn = SOAPpy.WSDL.Proxy("VirtualServer.wsdl")
names = conn.getVirtualServerNames()
enabled = conn.getEnabled(names)

for i in range(0,len(names)):
        if ( enabled[i] ):
                print names[i]
```

By default, most SOAP implementations read the location of the SOAP server from the WSDL file[4]. However, for security reasons, the location of the Zeus Admin Server (including the required administrator username and password) is not embedded in the Zeus WSDL files.

Most SOAP toolkits allow you to override the location specified in the WSDL file, but Python's SOAP.py module does not. Before you run this example, edit your WSDL files. Look for the soap:address node at the very end of each WSDL file and edit appropriately:

```
<service name="VirtualServer">
   <port name="VirtualServerPort"
         binding="zeusns:VirtualServerBinding">
      <soap:address
         location="https://username:password@host:9090/soap" />
   </port>
</service>
```

Run the Python script as follows[5]:

```
$ ./listVS.py
```

[4] The WSDL interface specifications for the Zeus Control API are located in `ZEUSHOME/zxtm/etc/wsdl/`.

[5] This example was tested with Python 2.3.5 and version 0.11.5 of the SOAP.py library. Earlier versions of SOAP.py (0.8.4) could not correctly parse the WSDL file.

```
Main website
Mail servers
Test site
```

2.1.5 listVS.php using PHP 5

```
#!/usr/bin/php5

<?
$conn = new SoapClient( "VirtualServer.wsdl",
    array('login' => "username", 'password' => "password") );

$names = $conn->getVirtualServerNames();
$enabled = $conn->getEnabled($names);

for ($i=0; $i < count( $names ); $i++) {
  if ( $enabled[$i] )
    print "$names[$i]\n";
}
?>
```

You may need to enable the SOAP extensions in your `php.ini` file; follow the instructions at http://www.php.net/soap if necessary.

Like Python, the PHP Soap toolkit expects to find the location of the SOAP server in the WSDL file, and does not provide any means to override it. However, you can specify the login details from your application, so these do not need to be embedded in the WSDL:

```
<service name="VirtualServer">
   <port name="VirtualServerPort"
         binding="zeusns:VirtualServerBinding">
      <soap:address location="https://host:9090/soap" />
   </port>
</service>
```

Run the PHP script as follows

```
$ ./listVS.php
Main website
Mail servers
Test site
```

2.2 Fault Handling

The Zeus Control API uses standard SOAP fault handling to inform the client application of errors. The type of fault returned depends on the error that occurred; for instance an 'ObjectDoesNotExist' fault will be returned when trying to set a property for a Virtual Server that doesn't exist. Information contained inside the fault will help determine more information about the error.

In addition to the specific faults specified for the functions, applications should be written to handle generic failures for which a specific fault does not exist.

Fault handling differs depending on the API being used. You should refer to your API documentation for details on how best to handle faults.

The following examples show code snippets of how to handle faults with various standard libraries.

2.2.1 Fault handling with SOAP::Lite

As SOAP::Lite doesn't read the WSDL files, the fault handling code needs to process the fault structures manually:

```
# This is the url of the Zeus admin server
my $admin_server = 'https://<user>:<pass>@adminserver:9090';

my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap")
    -> on_fault( \&handle_fault );

sub handle_fault
{
   my( $soap, $res ) = @_;

   if( ! $res ) die "A transport error occured\n";

   if( ! ref $res ) die $res;

   # $res is a SOAP fault - extract the information in it
   if( $res->faultdetail ) {
      my $detail = $res->faultdetail;
      my @elems = keys %$detail;
      my $fault = $elems[0];

      my $msg = "SOAP Fault: $fault\n";

      # Extract out the components of the fault
```

```
      foreach my $key( qw( errmsg object key value ) ) {
         if( defined $detail->{$fault}->{$key} ) {
            $msg .= "  $key: " . $detail->{$fault}->{$key} ."\n";
         }
      }
     die $msg;
   } else {
      die "SOAP Fault: " . $res->faultcode . ": " .
         $res->faultstring . "\n";
   }
}

# Could throw 'ObjectDoesNotExist'
$conn->setEnabled( [ "my-virtual-server" ], [ 1 ] );
```

2.2.2 Fault handling using C sharp

Like perl, the fault detail structure needs to be inspected manually:

```
try {
   p.setEnabled( new string[] { "my-virtual-server" },
                 new bool[] { true } );
} catch( SoapException fault ) {
   string msg = "";
   // Look at the fault detail XML tree
   if( fault.Detail != null && fault.Detail.FirstChild != null ) {
      XmlNode detail = fault.Detail.FirstChild;

      // The SOAP fault is the name of the first child of the
      // fault detail
      msg += "SOAP Fault: " + detail.LocalName + "\n";

      // And the other members of the fault are children
      XmlNodeList children = detail.ChildNodes;
      for( int i = 0 ; i < children.Count ; i++ ) {
         msg += "  " + children[i].Name + ": " +
                       children[i].InnerText + "\n";
      }
   } else {
      // Otherwise this is a generic fault - just print the fault
      msg = fault.ToString();
   }
   Console.Write( msg );
```

```
}
```

2.2.3 Fault handling using Java

Fault handling is built into the Java AXIS libraries; the SOAP faults are translated into standard Java exceptions which make it very easy to handle faults:

```
VirtualServerLocator vsl = new VirtualServerLocator();
vsl.setVirtualServerPortEndpointAddress(
      "https://<user>:<pass>@adminserver:9090/soap" );
VirtualServerPort vsp = vsl.getVirtualServerPort();

try {
   vsp.setEnabled( new String[] { "my-virtual-server" },
                   new boolean[] { true } );

   System.out.println( "Virtual Server enabled successfully" );
} catch( ObjectDoesNotExist e ) {
   System.err.println(
      "Virtual Server '" + e.getObject() + "' does not exist" );
} catch( RemoteException e ) {
   System.err.println( "Generic exception: " + e );
}
```

Using Perl SOAP::Lite

Unlike most other SOAP APIs, Perl's SOAP::Lite does not take regard of the WSDL specification for the Control API interface. Special measures must be taken to use the Control API accurately.

3.1 Control API methods

A method can be invoked from a SOAP::Lite connection object that specifies the appropriate interface:

```
# Create a connection object that uses the VirtualServer interface
my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap");

# You can invoke any of the methods of the VirtualServer interface
my $res = $conn->getVirtualServerNames();
```

If you need to use several interfaces (for example, VirtualServer and Pool), you will need to construct a SOAP::Lite connection object for each.

If you attempt to invoke a method that does not exist for the interface, the method call will fail and the `on_fault` fault handler (if specified) will be called.

3.2 Control API Enumerations

An Enumeration is a particular datatype with a restricted, named set of values. For example, a Pool has a limited set of load balancing algorithms that are represented by the `Pool.LoadBalancingAlgorithm` enumeration:

The enumeration is defined as follows:

Pool.LoadBalancingAlgorithm

```
enum Pool.LoadBalancingAlgorithm {
   roundrobin,      # Round Robin
   wroundrobin,     # Weighted Round Robin
   cells,           # Perceptive
   connections,     # Least Connections
   wconnections,    # Weighted Least Connections
   responsetimes,   # Fastest Response Time
   random           # Random node
}
```

The Pool interface contains two methods that use that enumeration:

getLoadBalancingAlgorithm(names)

Get the load balancing algorithms that each of the named pools uses.

```
Pool.LoadBalancingAlgorithm[] getLoadBalancingAlgorithm(
   String[] names
)
```

setLoadBalancingAlgorithm(names, values)

Set the load balancing algorithms that each of the named pools uses.

```
void setLoadBalancingAlgorithm(
   String[] names
   Pool.LoadBalancingAlgorithm[] values
)
```

Perl's SOAP::Lite library correctly encodes enumerations in SOAP requests, so you can use them in a literal fashion:

```
$conn->setLoadBalancingAlgorithm( [ $poolName ], ['connections'] );
```

In a SOAP response, you need to provide a custom Deserializer so that the SOAP::Lite library can convert the values in the SOAP response into appropriate internal representations (i.e. literal strings):

```
BEGIN {
   package MyDeserializer;
   @MyDeserializer::ISA = 'SOAP::Deserializer';

   sub typecast {
      my( $self, $val, $name, $attrs, $children, $type ) = @_;
      if( $type && $type =~ m@http://soap.zeus.com/zxtm/@ ) {
         return $val;
      }
      return undef;
   };
}

my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/Pool/')
    -> proxy("$admin_server/soap")
    -> deserializer( MyDeserializer->new );
```

The following code sample illustrates how to use Control API methods that use Enumerations:

```
#!/usr/bin/perl -w

use SOAP::Lite 0.6;

# Provide our own Deserializer to deserialize enums correctly
BEGIN {
   package MyDeserializer;
   @MyDeserializer::ISA = 'SOAP::Deserializer';

   sub typecast {
      my( $self, $val, $name, $attrs, $children, $type ) = @_;
      if( $type && $type =~ m@http://soap.zeus.com/zxtm/@ ) {
         return $val;
      }
      return undef;
   };
}

# This is the url of the Zeus admin server
my $admin_server = 'https://username:password@host:9090';

# The pool to edit
my $poolName = $ARGV[0] or die "No pool specified";

my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/Pool/')
    -> proxy("$admin_server/soap")
    -> deserializer( MyDeserializer->new )
    -> on_fault( sub  {
         my( $conn, $res ) = @_;
         die ref $res?$res->faultstring:$conn->transport->status; }
);

# Get the load balancing algorithm
my $res = $conn->getLoadBalancingAlgorithm( [ $poolName ] );
my $alg = @{$res->result}[0];
print "Pool $poolName uses load balancing algorithm $alg\n";

# Change the algorithm to least connections, and check it worked
$conn->setLoadBalancingAlgorithm( [ $poolName ], ['connections'] );
```

```
$res = $conn->getLoadBalancingAlgorithm( [ $poolName ] );
print "Algorithm has been changed to @{$res->result}[0]\n";

# Now change it back again
$conn->setLoadBalancingAlgorithm( [ $poolName ], [ $alg ] );
$res = $conn->getLoadBalancingAlgorithm( [ $poolName ] );
print "Algorithm changed back to @{$res->result}[0]\n";
```

3.3 Control API Structures

A Structure is a complex datatype that contains several parameters. For example, the key configuration settings for a Virtual Server are represented by a `VirtualServer.BasicInfo` structure that defines the port, protocol and default pool for that Virtual Server:

VirtualServer.BasicInfo

This structure contains the basic information for a virtual server. It is used when creating a server, or modifying the port, protocol or default pool of a server.

```
struct VirtualServer.BasicInfo {
   # The port to listen for incoming connections on.
   Integer port;

   # The protocol that this virtual server handles.
   VirtualServer.Protocol protocol;

   # The default pool that traffic to this virtual server will go
   # to.
   String default_pool;
}
```

This structure contains three elements; an Integer (the port number), an Enumeration (`VirtualServer.Protocol` – the protocol) and a string (the name of the default pool).

The method `VirtualServer.addVirtualServer()` takes a `VirtualServer.BasicInfo` structure which can be constructed as follows:

```
my $basicInfo = {
   port         => '443',
   protocol     => 'https',
   default_pool => 'Server Pool 1'
};

$res = $conn->addVirtualServer( [ $vsName ], [ $basicInfo ] );
```

If you call the method `VirtualServer.getBasicInfo()`, it will return a corresponding array of `VirtualServer.BasicInfo` structures that can be unpacked as follows:

```
$res = $conn->getBasicInfo( [ $vsName ] );
my $r = @{$res->result}[0];

print "Virtual Server $vsName:\n";
print "   port $r->{port}, protocol $r->{protocol}, " .
      "pool $r->{default_pool}\n";
```

The following code sample illustrates how to create a virtual server and manage the BasicInfo structure:

```
#!/usr/bin/perl -w

use SOAP::Lite 0.6;

# Provide our own Deserializer so to deserialize enums correctly
BEGIN {
   package MyDeserializer;
   @MyDeserializer::ISA = 'SOAP::Deserializer';

   sub typecast {
      my( $self, $val, $name, $attrs, $children, $type ) = @_;
      if( $type && $type =~ m@http://soap.zeus.com/zxtm/@) {
         return $val;
      }
      return undef;
   };
}

# This is the url of the Zeus admin server
my $admin_server = 'https://user:password@hostname:9090';

# The virtual server to create
my $vsName = $ARGV[0] or die "No vs specified";

my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap")
    -> deserializer( MyDeserializer->new )
    -> on_fault( sub  {
         my( $conn, $res ) = @_;
```

```
        die ref $res?$res->faultstring:$conn->transport->status; }
);

# Construct the basic info structure
my $basicInfo = {
   port         => '443',
   protocol     => 'https',
   default_pool => 'discard'
};

$res = $conn->addVirtualServer( [ $vsName ], [ $basicInfo ] );

$res = $conn->getBasicInfo( [ $vsName ] );
my $r = @{$res->result}[0];

print "Virtual Server $vsName:\n";
print "   port $r->{port}, protocol $r->{protocol}, " .
      "pool $r->{default_pool}\n";
```

Sample Control API applications

The Zeus Control API can perform almost any configuration task that can be accomplished using the Zeus Admin Server. Its strength comes from how it can be driven by other management applications elsewhere in the network.

4.1 Blocking traffic from an IP address

An Intrusion Detection System (IDS) or a live log analysis tool may identify remote hosts which are sending undesired traffic – malicious requests, port scans, or simply excessive numbers of requests in an attempt to mount a denial-of-service attack.

The IDS may be located behind the traffic manager cluster, for example, if it needs to inspect SSL traffic that has been decrypted by the traffic managers. In this case, the IDS can use the Control API to update the traffic manager cluster to prevent it from accepting any more traffic from the suspected IP address.

The following Zeus Control API application modifies a named Service Protection Policy, adding an IP address to the list of banned IP addresses. The Service Protection Policy should be assigned to the appropriate Virtual Servers managing traffic in the cluster.

4.1.1 Perl Example

```
#!/usr/bin/perl -w

use SOAP::Lite 0.60;

# This is the url of the Zeus admin server
my $admin_server = 'https://username:password@host:9090';

# The protection policy to edit, and the node to add
my $name = "My protection class";
my $badIP  = "10.100.1.10";

my $conn = SOAP::Lite
    -> uri('http://soap.zeus.com/zxtm/1.0/Catalog/Protection/')
    -> proxy("$admin_server/soap")
    -> on_fault( sub  {
         my( $conn, $res ) = @_;
         die ref $res ? $res->faultstring :
             $conn->transport->status; } );

$conn->addBannedAddresses( [ $name ], [ [ $badIP ] ] );
```

Notes

We are accessing the 'Catalog/Protection' uri to edit a service protection class. With a WSDL-based interface, we'd use the Catalog.Protection.wsdl interface.

We use the addBannedAddresses() function. Like most Control API functions, this takes a series of arrays as arguments:

1. A list of service protection policies
2. A list of lists of banned IP addresses

This means that the function can perform bulk updates, modifying several objects simultaneously.

This example includes a basic on_fault handler which is called if an error ever occurs. The handler will report a transport error if the SOAP application could not connect to the remote SOAP server. Otherwise, it will report a SOAP error.

For more a more sophisticated example of a Perl fault handler, refer to section 2.2.1.

4.1.2 C# Example

```
using System;
using System.Net;
using System.IO;
using System.Security.Cryptography.X509Certificates;

public class AllowSelfSignedCerts : ICertificatePolicy {
   public bool CheckValidationResult(
         ServicePoint sp, X509Certificate cert,
         WebRequest request, int problem )
   { return true; }
}

public class addBannedAddress {

   public static void Main( string [] args )
   {
      System.Net.ServicePointManager.CertificatePolicy =
         new AllowSelfSignedCerts();

      string url= "https://host:9090/soap";
      string username = "username";
      string password = "password";

      string name = "My protection class";
      string badIP = "10.100.1.10";
```

```
      try {
         Zeus.CatalogProtection p =
                 new Zeus.CatalogProtection();
         p.Url = url;
         p.Credentials = new NetworkCredential(
            username, password );

         p.addBannedAddresses( new string[] { name },
             new string[][] { new string[]{ badIP } } );
      } catch ( Exception e ) {
         Console.WriteLine( "{0}", e );
      }
   }
}
```

4.2 Adding a node to a pool

Provisioning systems can dynamically deploy applications across servers, perhaps in reaction to increased server load. This example demonstrates a Control API application that modifies the nodes that a pool balances traffic to.

If the pool is using the 'Perceptive' algorithm, then load will slowly be ramped up on newly introduced nodes, gauging their potential performance, until they run at the same speed as the other nodes in the pool. This 'Slow Start' capability ensures that new nodes are not immediately overloaded with a large burst of traffic.

4.2.1 Perl Example

```
#!/usr/bin/perl -w

use SOAP::Lite 0.60;

# This is the url of the Zeus admin server
my $admin_server = 'https://username:password@host:9090';

# The pool to edit, and the node to add
my $poolName = "test pool";
my $newNode  = "10.100.1.10:80";

my $conn = SOAP::Lite
    -> uri('http://soap.zeus.com/zxtm/1.0/Pool/')
    -> proxy("$admin_server/soap")
    -> on_fault( sub  {
```

```
         my( $conn, $res ) = @_;
         die ref $res ? $res->faultstring :
             $conn->transport->status; } );

# Get a list of pools
my $res = $conn->getPoolNames();
my @names = @{$res->result};

# Get the nodes for each pool
$res = $conn->getNodes( \@names );

# Build a hash %nodes: pool->[ node list ]
my %nodes;
@nodes{@names} = @{$res->result};

if( !defined $nodes{$poolName} ) {
   die "Pool $poolName does not exist...";
}

if( grep { $_ eq $newNode } @{$nodes{$poolName}} ) {
   die "Pool $poolName already contains $newNode";
}

# Add one node to the pool
$res = $conn->addNodes( [ $poolName ], [ [ $newNode ] ] );

# We're done!  Verify that the node has been added
$res = $conn->getNodes( [ $poolName ] );
my @newnodes = @{${$res->result}[0]};

my $expected = join " ",
   sort @{$nodes{$poolName}}, $newNode;
my $actual   = join " ", sort @newnodes;

if( $expected ne $actual ) {
   die "New node list is '$actual'; expected '$expected'";
}
```

Notes

This example uses careful error checking to make sure that the Control API methods are not called incorrectly. For example, if a method tries to add a node to a pool that did not exist, a SOAP fault will be raised. Perl's `on_fault` handler will be called if this happens.

The example illustrates Perl's hash slice technique to quickly build an associative array, mapping pool name to a list of nodes:

```
my $res = $conn->getPoolNames();
my @names = @{$res->result};
$res = $conn->getNodes( \@names );
my %nodes;
@nodes{@names} = @{$res->result};
```

This is a very easy way to take advantage of the fact that the Zeus Control API methods are all bulk-enabled, i.e., they are designed to process lists of objects efficiently.

The listVS example could also use a hash slice, as follows:

```
my $res = $conn->getVirtualServerNames();
my @names = @{$res->result};
$res = $conn->getEnabled( \@names );
my %enabled;
@enabled{@names} = @{$res->result};
```

A Zeus Control API application could update the configuration by modifying the hash:

```
# Turn everything off...
foreach my $name( keys %enabled ) {
   $enabled{ $name } = 0;
}
```

It could then bulk-commit the new configuration with a single method call:

```
$res = $conn->setEnabled(
            [ keys %enabled ], [ values %enabled ] );
```

4.2.2 C# Example

```
using System;
using System.Net;
using System.IO;
using System.Security.Cryptography.X509Certificates;

public class AllowSelfSignedCerts : ICertificatePolicy {
   public bool CheckValidationResult(
         ServicePoint sp, X509Certificate cert,
         WebRequest request, int problem )
   {
      return true;
   }
}
public class addNode {
   public static void Main( string [] args )
   {
      System.Net.ServicePointManager.CertificatePolicy =
         new AllowSelfSignedCerts();

      string url= "https://host:9090/soap";
      string username = "username";
      string password = "password";

      string poolName = "test pool";
      string newNode  = "10.100.1.10:80";

      try {
         Zeus.Pool p = new Zeus.Pool();
         p.Url = url;
         p.Credentials = new NetworkCredential(
            username, password );

         string[] names = p.getPoolNames();
         string[][] allnodes = p.getNodes( names );

         string[] nodes = new string[]{};
         bool found = false;
         for( int i = 0 ; i < names.Length ; i++ ) {

            if( names[i] == poolName ) {
               nodes = allnodes[i];
               found = true;
```

```
                break;
            }
         }

         if( ! found ) {
            Console.WriteLine( "Pool {0} doesn't exist", poolName );
            Environment.Exit( 1 );
         }

         found = false;
         for( int i = 0 ; i < nodes.Length ; i++ ) {
            if( nodes[i] == newNode ) {
               found = true;
            }
         }
         if( found ) {
            Console.WriteLine( "Pool {0} already contains {1}",
               poolName, newNode );
            Environment.Exit( 1 );
         }

         // Add one node to the pool
         p.addNodes( new string[] { poolName },
            new string[][] { new string[] { newNode } } );

      } catch ( Exception e ) {
         Console.WriteLine( "{0}", e );
      }
   }
}
```

4.3 Reconfiguring your site based on Traffic Load

In this example, we'll monitor the performance of the JSP pages on a website. If the performance drops below an acceptable level, we'll use the Control API to enable a TrafficScript rule that prevents more users from logging into the website. Once performance climbs back to an acceptable level, the Control API application can disable the rule.

The TrafficScript Rule

To prevent users from logging into the site, we could use a TrafficScript rule similar to the following:

```
$path = http.getPath();
if( string.endsWith( $path, "login.jsp" ) ) {
   http.redirect( "/content/login_disabled.html" );
}
```

Add this rule to the Rules catalog, calling it 'Disable Login'. Configure it as a request rule for your virtual server, but set it to be disabled. The Zeus Control API application will use the Virtual Server `getRules()` and `setRules()` functions to modify the 'enabled' status of the rule.

Monitoring Performance

Performance of the web application can be monitored in a variety of ways:

- Using statistics gathered from the web application itself.
- Using an external monitoring tool to send probe requests.
- Monitoring the node response times using SNMP and the Service Level Monitoring capability.
- Using the SNMP or email alerts raised by Service Level Monitoring to drive the Control API applications directly.

Enabling and Disabling the Rule

The following Control API code retrieves the list of response rules that the named virtual server is using. It searches for the rule named 'Disable Login' and enables it. If the rule is not present, it adds it as the first rule to be executed.

4.3.1 Perl example

```
#!/usr/bin/perl -w

use SOAP::Lite 0.60;

# Provide our own Deserializer so to deserialize enums correctly
BEGIN {
   package MyDeserializer;
   @MyDeserializer::ISA = 'SOAP::Deserializer';

   sub typecast {
      my( $self, $val, $name, $attrs, $children, $type ) = @_;
```

```
      if( $type && $type =~ m@http://soap.zeus.com/zxtm/@) {
         return $val;
      }
      return undef;
   };
}

# This is the url of the Zeus admin server
my $admin_server = 'https://username:password@host:9090';

# The virtual server to edit, and the rule to enable
my $vsName = "Main web site";
my $rule   = "Disable Login";

my $conn = SOAP::Lite
    -> uri('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap")
    -> deserializer( MyDeserializer->new )
    -> on_fault( sub  {
      my( $conn, $res ) = @_;
      die ref $res ? $res->faultstring:conn->transport->status; } );

# Get a list of rules used by the named virtual server
my $res = $conn->getRules( [ $vsName ] );
my @rules = @{${$res->result}[0]};

my $found = 0;
foreach my $f ( @rules ) {
   if( $f->{name} eq $rule ) {
      $f->{enabled} = 1; $found = 1; last;
   }
}
if( !$found ) {
   # Add a new rule to the start of the list
   unshift @rules, {
      name => $rule,
      enabled => 1,
      run_frequency => 'only_first'
   };
}

$res = $conn->setRules( [ $vsName ], [ [ @rules ] ] );
```

Notes

The rules manipulation functions take a compound `VirtualServer.Rule` structure which uses an enumeration named '`RuleRunFlag`'.

The `SOAP::Lite` interface presents this structure as a standard Perl hash, requiring a Deserializer to manage the enumeration.

C# presents it as an object of type `Zeus.VirtualServerRule` containing an enumeration of type `Zeus.VirtualServerRuleRunFlag`.

4.3.2 C# Example

```
using System;
using System.Net;
using System.IO;
using System.Security.Cryptography.X509Certificates;

public class AllowSelfSignedCerts : ICertificatePolicy {
   public bool CheckValidationResult(
         ServicePoint sp, X509Certificate cert,
         WebRequest request, int problem )
   {
      return true;
   }
}

public class addNode {
   public static void Main( string [] args )
   {
      System.Net.ServicePointManager.CertificatePolicy =
         new AllowSelfSignedCerts();

      string url= "https://host:9090/soap";
      string username = "username";
      string password = "password";

      string vsName  = "Main web site";
      string rule    = "Disable login";
      try {
         Zeus.VirtualServer p = new Zeus.VirtualServer();
         p.Url = url;
         p.Credentials = new NetworkCredential(
            username, password );

         // Get a list of rules used by the named
```

```
         // virtual server
         Zeus.VirtualServerRule[][] rules =
            p.getRules( new string[] { vsName } );

         // Search for the rule to enable
         bool found = false;
         foreach( Zeus.VirtualServerRule r in rules[0] ) {
            if( r.name == rule ) {
               found = true; r.enabled = true;
            }
         }
         if( ! found ) {
            // Add a new rule to the start of the list
            Zeus.VirtualServerRule[] newrules = new
               Zeus.VirtualServerRule[rules[0].Length + 1];
            newrules[0] = new Zeus.VirtualServerRule();
            newrules[0].name = rule;
            newrules[0].enabled = true;
            newrules[0].run_frequency =
               Zeus.VirtualServerRuleRunFlag.only_first;

            Array.Copy( rules[0], 0, newrules, 1, rules[0].Length );

            rules[0] = newrules;
         }
         p.setRules( new string[] { vsName }, rules );

      } catch ( Exception e ) {
         Console.WriteLine( "{0}", e );
      }
   }
}
```

Troubleshooting

5.1 Overview

Building Zeus Control API applications is an involved process. This chapter lists some useful techniques.

5.1.1 Can't find the WSDL files?

The WSDL interface specifications are located in:

```
ZEUSHOME/zxtm/etc/wsdl/
```

`ZEUSHOME` is the installation directory for your Zeus software, typically `/usr/local/zeus/`.

You can also download them from the SOAP API page in the online help.

5.2 General Debugging Techniques

5.2.1 Log Files

In the event of a problem, review the following error logs:

- Zeus Software: `ZEUSHOME/log/errors`

  Validation errors and incorrect configuration problems will be reported in this log file.

- Zeus Admin Server: `ZEUSHOME/admin/log/errors`

  The Admin Server processes the SOAP requests and sends the new configuration to the Zeus software. Any SOAP protocol or transport errors will be reported here.

5.2.2 Snooping the SOAP traffic

Your Zeus Control API application will send SOAP requests to the Zeus Admin server, which typically listens using SSL on port 9090. If your SOAP toolkit does not support debugging or tracing, you can use a network snooping tool such as WireShark[6] to inspect the SOAP request and response, to verify that the request is sent correctly, and that the response does not contain any errors messages that are not reported by your application's interface code.

Of course, if your SOAP transaction is encrypted with SSL, it will not be easy to inspect it:

1. Disable SSL on the Zeus Admin Server as follows:

[6] http://www.wireshark.org/ WireShark is available for all major platforms, including Windows, Linux and Mac OS X.

a. Edit the file `ZEUSHOME/admin/website`, and comment out the line 'security!enabled yes' by prefixing it with a '#':

```
#security!enabled yes
```

b. Restart the software[7]:

```
# /usr/local/zeus/restart-zeus
```

2. Modify your Zeus Control API application to use an 'http://' url rather than an 'https://' one.

5.3 Debugging with Perl

5.3.1 Problems with WSDL interfaces

Because Perl's SOAP::Lite does not make explicit reference to the WSDL interface specification, it can be easy to make errors which are only detected at run time.

Ensure that the URI in the SOAP::Lite objects you construct in your application is correct. Refer to the URI tables in chapter 6.

For example, to reference methods in the VirtualServer interface, you should use the following URI:

```
http://soap.zeus.com/zxtm/1.0/VirtualServer/
```

For methods in the Service Protection catalog, use the following URI:

```
http://soap.zeus.com/zxtm/1.0/Catalogs/Protection/
```

5.3.2 Using a Fault Handler

Your Perl SOAP::Lite client application can determine whether server or transport errors have occurred by inspecting the SOAP Fault that is raised on an error.

Section 2.2 covers Fault Handlers in detail.

[7] Alternatively, you can restart just the admin server component as follows:

```
# export ZEUSHOME=/usr/local/zeus
# $ZEUSHOME/admin/rc restart
```

Set ZEUSHOME to the correct installation directory for your configuration.

5.3.3 Recent SOAP::Lite versions

Versions of SOAP::Lite on or after 0.65_5 have a slightly different interface. When creating a SOAP::Lite connection, you should use a new 'ns' method instead of the previous 'uri' method:

```
# Versions prior to 0.65_5
my $conn = SOAP::Lite
    -> uri('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap");
```

```
# Versions 0.65_5 and later
my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/VirtualServer/')
    -> proxy("$admin_server/soap");
```

5.3.4 Perl deserializer Example

The SOAP::Lite module does not make use of Zeus's WSDL specification and so does not know how to deserialize some enumerations used. See chapter 3 for an example of this problem.

5.3.5 Tracing

When you import the SOAP::Lite module, use the following:

```
use SOAP::Lite 0.6 +trace => 'debug';
```

This will cause the SOAP::Lite module to output large amounts of debugging information.

The XML-messages you see after enabling the tracing are usually not formatted. To make them easier to read, set the readable flag on your connections:

```
my $conn = SOAP::Lite;

$conn->readable(1);
```

This will make the messages sent by the client more readable, but will not affect messages received from the server.

5.4 Debugging with C#

5.4.1 Fault Handlers

Pleae refer to section 2.2 for details on how to inspect any SOAP Faults that are raised as a result of a server or transport error.

5.4.2 Permissions Problems

The .NET environment enforces stringent security checks by default.

For example, by default, your Control API application cannot generate an HTTP request to a foreign site (such as the Zeus Admin Server) unless the application is running from a 'trusted location'. Remote filesystems which are locally mounted are untrusted, whereas local filesystems are trusted.

The location of your Control API application may affect whether it functions correctly or not.

5.5 Debugging with Java

5.5.1 Fault Handlers

Pleae refer to section 2.2 for details on how to inspect any SOAP Faults that are raised as a result of a server or transport error.

5.5.2 Tracing

For full SOAP tracing, you can run your Zeus Control API application as follows:

```
$ java -Djavax.net.debug=all listVS
```

Alternatively, you can enable debugging within your application:

```
System.setProperty( "javax.net.debug", "all" );
```

Function Reference

6.1 About the Zeus Control API functions

Zeus Control API functions generally operate on lists of configurations. For example, the VirtualServer `getEnabled()` function takes a list of virtual server names as its first argument, and returns a list of Boolean values, one for each named virtual server.

Some functions depend on compound structures for their arguments, and enumerated types are used to represent some configuration settings.

All of the methods, structures and enumerated types are specified in the WSDL interface files[8].

The `addRules()` and `getRules()` VirtualServer functions are good examples:

```
VirtualServer.Rule[][] getRules(
   String[] names
)

void addRules(
   String[] names
   VirtualServer.Rule[][] rules
)
```

Method prototypes for the VirtualServer getRules and addRules methods

[8] The WSDL interface specifications are located in `ZEUSHOME/zxtm/etc/wsdl/`

`getRules()` takes a list of virtual server names, returning a list of `VirtualServer.Rule` arrays:

```
struct VirtualServer.Rule {
   # The name of the rule.
   String name;

   # Whether the rule is enabled or not.
   Boolean enabled;

   # Whether the rule runs on every request response,
   # or just the first
   VirtualServer.RuleRunFlag run_frequency;
}
```

Definition of the VirtualServer.Rule structure

The VirtualServer.Rule structure includes an enumerated type:

```
enum VirtualServer.RuleRunFlag {
   # Run on every request or response
   run_every,

   # Run only on the first request or response
   only_first
}
```

Definition of the VirtualServer.RuleRunFlag enumerated type

Your SOAP toolkit will represent these WSDL methods, structures and enumerated types in a form appropriate for the language in use:

- Perl uses methods in the SOAP::Lite object. Structures map straightforwardly onto Perl associative arrays. You need to provide an explicit deserializer to typecast enumerated type values into string values. See chapter 3 for details.
- The C# and Java toolkits provide a means to convert the WSDL files into C# or Java source files, with fully typed classes, structures and enumerations to represent the SOAP methods, structures and enumerated types.

Take a look at section 4.3 for a worked example that illustrates the use of the methods, structures and enumerations.

6.2 VirtualServer

URI: `http://soap.zeus.com/zxtm/1.0/VirtualServer/`

The VirtualServer interface allows management of Virtual Server objects. Using this interface, you can create, delete and rename virtual server objects, and manage their configuration.

6.2.1 Methods

addCompressionMIMETypes(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

For each named virtual server, add new MIME types to the list of types to compress.

```
void addCompressionMIMETypes(
   String[] names
   String[][] values
)
```

addResponseRules(names, rules) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add new rules to be run on server responses for each of the named virtual servers. New rules are run after existing rules. If any of the rules are already configured to run, then they are enabled and flags are set to the values passed in.

```
void addResponseRules(
   String[] names
   VirtualServer.Rule[][] rules
)
```

addRules(names, rules) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add new rules to be run on client requests for each of the named virtual servers. New rules are run after existing rules. If any of the rules are already configured to run, then they are enabled and flags are set to the values passed in.

```
void addRules(
   String[] names
   VirtualServer.Rule[][] rules
)
```

addSSLClientCertificateAuthorities(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add new certificate authorities for validating client certificates for each of the named virtual servers.

```
void addSSLClientCertificateAuthorities(
   String[] names
   String[][] values
```

```
)
```

addSSLSites(names, ssl_sites) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Adds the specified SSLSite objects to the named virtual servers. These objects are mappings between destination addresses and the certificate used for SSL decryption those addresses. Each certificate is the name of an item in the SSL Certificates Catalog.

```
void addSSLSites(
   String[] names
   VirtualServer.SSLSite[][] ssl_sites
)
```

addVirtualServer(names, info) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidInput

Add each virtual servers, using the provided BasicInfo.

```
void addVirtualServer(
   String[] names
   VirtualServer.BasicInfo[] info
)
```

copyVirtualServer(names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError

Rename each of the named virtual servers.

```
void copyVirtualServer(
   String[] names
   String[] new_names
)
```

deleteSSLSites(names, site_ips) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Deletes the SSLSite objects that act on the IP addresses in the site_ips array for each of the named virtual servers. These objects are mappings between destination addresses and the certificate used for SSL decryption those addresses. Each certificate is the name of an item in the SSL Certificates Catalog.

```
void deleteSSLSites(
   String[] names
   String[][] site_ips
)
```

deleteVirtualServer(names) throws ObjectDoesNotExist, DeploymentError

Delete each of the named virtual servers.

```
void deleteVirtualServer(
   String[] names
)
```

editSSLSites(names, site_ips, ssl_sites) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Edits the SSLSite objects that act on the IP addresses in the site_ips array for each of the named virtual servers. These objects are mappings between destination addresses and the certificate used for SSL decryption those addresses. Each certificate is the name of an item in the SSL Certificates Catalog.

```
void editSSLSites(
   String[] names
   String[][] site_ips
   VirtualServer.SSLSite[][] ssl_sites
)
```

getAddClusterClientIPHeader(names) throws ObjectDoesNotExist

Get whether a 'X-Cluster-Client-Ip' header should be added to each HTTP request, for each of the named virtual servers. The 'X-Cluster-Client-Ip' header contains the client's IP address.

```
Boolean[] getAddClusterClientIPHeader(
   String[] names
)
```

getBandwidthClass(names) throws ObjectDoesNotExist

Get the Bandwidth Class that each of the named virtual servers uses.

```
String[] getBandwidthClass(
   String[] names
)
```

getBasicInfo(names) throws ObjectDoesNotExist

Get the basic information for each of the named virtual servers. This information includes the port, the protocol the virtual server handles and the default pool for the traffic.

```
VirtualServer.BasicInfo[] getBasicInfo(
   String[] names
)
```

getCompressUnknownSize(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should compress documents with no given size.

```
Boolean[] getCompressUnknownSize(
   String[] names
)
```

getCompressionEnabled(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should compress web pages before sending to the client.

```
Boolean[] getCompressionEnabled(
   String[] names
)
```

getCompressionLevel(names) throws ObjectDoesNotExist

Get the gzip compression level, for each of the named virtual servers.

```
Unsigned Integer[] getCompressionLevel(
   String[] names
)
```

getCompressionMIMETypes(names) throws ObjectDoesNotExist

Get the list of MIME types to compress, for each of the named virtual servers.

```
String[][] getCompressionMIMETypes(
   String[] names
)
```

getCompressionMaxSize(names) throws ObjectDoesNotExist

Get the maximum document size to compress, in bytes, for each of the named virtual servers. A document size of '0' means 'unlimited'.

```
Unsigned Integer[] getCompressionMaxSize(
   String[] names
)
```

getCompressionMinSize(names) throws ObjectDoesNotExist

Get the minimum document size to compress, in bytes, for each of the named virtual servers.

```
Unsigned Integer[] getCompressionMinSize(
```

```
   String[] names
)
```

getConnectTimeout(names) throws ObjectDoesNotExist

Get the time to wait for data from a new connection, in seconds, for each of the named virtual servers. If no data is received in this time, the connection will be closed.

```
Unsigned Integer[] getConnectTimeout(
   String[] names
)
```

getCookieDomainRewriteMode(names) throws ObjectDoesNotExist

Get how each of the named virtual servers should rewrite the domain portion of cookies set by a back-end web server.

```
VirtualServer.CookieDomainRewriteMode[] getCookieDomainRewriteMode(
   String[] names
)
```

getCookieNamedDomain(names) throws ObjectDoesNotExist

Get the domain to use when rewriting cookie domains, for each of the named virtual servers.

```
String[] getCookieNamedDomain(
   String[] names
)
```

getCookiePathRewrite(names) throws ObjectDoesNotExist

For each of the named virtual servers, get the regex and replacement for rewriting the path portion of a cookie.

```
VirtualServer.RegexReplacement[] getCookiePathRewrite(
   String[] names
)
```

getCookieSecureFlagRewriteMode(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should modify the 'secure' tag of cookies set by a back-end web server.

```
VirtualServer.CookieSecureFlagRewriteMode[]
getCookieSecureFlagRewriteMode(
   String[] names
)
```

getDefaultPool(names) throws ObjectDoesNotExist

Get the default Pool that traffic is sent to for each of the named virtual servers.

```
String[] getDefaultPool(
   String[] names
)
```

getEnabled(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers is enabled (i.e. serving traffic).

```
Boolean[] getEnabled(
   String[] names
)
```

getErrorFile(names) throws ObjectDoesNotExist

Get the file names of the error texts that each of the named virtual servers will send back to a client in case of back-end or internal errors.

```
String[] getErrorFile(
   String[] names
)
```

getFTPDataSourcePort(names) throws ObjectDoesNotExist

Get the source port each of the named virtual servers should use for active-mode FTP data connections. If 0, a random high port will be used, otherwise the specified port will be used. If a port below 1024 is required you must first explicitly permit use of low ports with the ftp_data_bind_low global setting.

```
Unsigned Integer[] getFTPDataSourcePort(
   String[] names
)
```

getFTPForceClientSecure(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should require incoming FTP data connections (from clients) to originate from the same IP address as the corresponding control connection.

```
Boolean[] getFTPForceClientSecure(
   String[] names
)
```

getFTPForceServerSecure(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should require incoming FTP data connections (from nodes) to originate from the same IP address as the corresponding control connection.

```
Boolean[] getFTPForceServerSecure(
   String[] names
)
```

getFTPPortRange(names) throws ObjectDoesNotExist

Get the port range used for FTP data connections for each of the named virtual servers.

```
VirtualServer.FTPPortRange[] getFTPPortRange(
   String[] names
)
```

getFTPSSLData(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should use SSL on the data connection as well as the control connection

```
Boolean[] getFTPSSLData(
   String[] names
)
```

getKeepalive(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should allow clients to maintain keepalive connections.

```
Boolean[] getKeepalive(
   String[] names
)
```

getKeepaliveTimeout(names) throws ObjectDoesNotExist

Get the time that an idle keepalive connection should be kept open for, in seconds, for each of the named virtual servers.

```
Unsigned Integer[] getKeepaliveTimeout(
   String[] names
)
```

getListenAddresses(names) throws ObjectDoesNotExist

Get the specific IP addresses and hostnames that each of the named virtual servers are listening on. This will return an empty array for a virtual server if it is listening on all addresses.

```
String[][] getListenAddresses(
   String[] names
)
```

getListenOnAllAddresses(names) throws ObjectDoesNotExist

For each of the named virtual servers, gets whether the virtual server is listening on all IP addresses

```
Boolean[] getListenOnAllAddresses(
   String[] names
)
```

getListenTrafficIPGroups(names) throws ObjectDoesNotExist

Get the specific Traffic IP Groups that each named virtual server listens on. This will return an empty array for a virtual server if it is listening on all addresses.

```
String[][] getListenTrafficIPGroups(
   String[] names
)
```

getLocationDefaultRewriteMode(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should rewrite the 'Location' header. The rewrite is only performed if the location rewrite regex didn't match.

```
VirtualServer.LocationDefaultRewriteMode[]
getLocationDefaultRewriteMode(
   String[] names
)
```

getLocationRewrite(names) throws ObjectDoesNotExist

For each of the named virtual servers, get the regex, and replacement for rewriting any 'Location' headers.

```
VirtualServer.RegexReplacement[] getLocationRewrite(
   String[] names
)
```

getLogClientConnectionFailures(names) throws ObjectDoesNotExist

Get whether the virtual server will log client connection failures.

```
Boolean[] getLogClientConnectionFailures(
   String[] names
)
```

getLogEnabled(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should log each connection to a disk on the file system.

```
Boolean[] getLogEnabled(
   String[] names
)
```

getLogFilename(names) throws ObjectDoesNotExist

Get the name of the file used to store request logs, for each of the named virtual servers.

```
String[] getLogFilename(
   String[] names
)
```

getLogFormat(names) throws ObjectDoesNotExist

Get the log file format for each of the named virtual servers.

```
String[] getLogFormat(
   String[] names
)
```

getLogServerConnectionFailures(names) throws ObjectDoesNotExist

Get whether the virtual server will log server connection failures.

```
Boolean[] getLogServerConnectionFailures(
   String[] names
)
```

getMIMEAutoDetect(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should auto-detect MIME types if the server does not provide them.

```
Boolean[] getMIMEAutoDetect(
   String[] names
)
```

getMIMEDefaultType(names) throws ObjectDoesNotExist

Get the MIME type that the server uses as its 'default', for each of the named virtual servers. Responses with this mime type will be auto-corrected by the virtual server if this setting is enabled.

```
String[] getMIMEDefaultType(
   String[] names
)
```

getMaxClientBuffer(names) throws ObjectDoesNotExist

Get the amount of memory used to store data sent by the client, in bytes, for each of the named virtual servers.

```
Unsigned Integer[] getMaxClientBuffer(
   String[] names
)
```

getMaxServerBuffer(names) throws ObjectDoesNotExist

Get the amount of memory used to store data returned by the server, in bytes, for each of the named virtual servers.

```
Unsigned Integer[] getMaxServerBuffer(
   String[] names
)
```

getNote(names) throws ObjectDoesNotExist

Get the note for each of the named virtual servers.

```
String[] getNote(
   String[] names
)
```

getPort(names) throws ObjectDoesNotExist

Get the port that each of the named virtual servers listens on for incoming connections.

```
Unsigned Integer[] getPort(
   String[] names
)
```

getProtection(names) throws ObjectDoesNotExist

Get the Service Protection Settings that are used to protect each of the named virtual servers.

```
String[] getProtection(
   String[] names
)
```

getProtocol(names) throws ObjectDoesNotExist

Get the protocol that each of the named virtual servers uses.

```
VirtualServer.Protocol[] getProtocol(
   String[] names
)
```

getProxyClose(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should send a FIN packet on to the back-end server when it is received from the client. The alternative is to close the connection to the client immediately. If your traffic manager is responding to the request itself, enabling this setting will cause your traffic manager to continue writing the response even after it has received a FIN from the client.

```
Boolean[] getProxyClose(
   String[] names
)
```

getRTSPPortRange(names) throws ObjectDoesNotExist

Get the port range used for RTSP streaming data connections, for each of the named virtual servers.

```
VirtualServer.PortRange[] getRTSPPortRange(
   String[] names
)
```

getRequestSyslogEnabled(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should log each connection to a remote syslog server.

```
Boolean[] getRequestSyslogEnabled(
   String[] names
)
```

getRequestSyslogFormat(names) throws ObjectDoesNotExist

Get the remote log line format for each of the named virtual servers.

```
String[] getRequestSyslogFormat(
   String[] names
)
```

getRequestSyslogIPEndpoint(names) throws ObjectDoesNotExist

Get the remote syslog endpoint for each of the named virtual servers

```
String[] getRequestSyslogIPEndpoint(
   String[] names
)
```

getResponseRules(names) throws ObjectDoesNotExist

Get the rules that are run on server responses for each of the named virtual servers.

```
VirtualServer.Rule[][] getResponseRules(
   String[] names
)
```

getRewriteSIPURI(names) throws ObjectDoesNotExist

Get whether the Request-URI of SIP requests will be replaced with the selected back-end node's address.

```
Boolean[] getRewriteSIPURI(
   String[] names
)
```

getRules(names) throws ObjectDoesNotExist

Get the rules that are run on client requests for each of the named virtual servers.

```
VirtualServer.Rule[][] getRules(
   String[] names
)
```

getSIPDangerousRequestMode(names) throws ObjectDoesNotExist

Get what should be done with requests that contain body data and should be routed to an external IP.

```
VirtualServer.SIPDangerousRequestMode[] getSIPDangerousRequestMode(
   String[] names
)
```

getSIPMaxConnectionMemory(names) throws ObjectDoesNotExist

Get maximum memory per connection.

```
Unsigned Integer[] getSIPMaxConnectionMemory(
   String[] names
)
```

getSIPMode(names) throws ObjectDoesNotExist

Get which mode of operation the SIP virtual server should run in.

```
VirtualServer.SIPMode[] getSIPMode(
   String[] names
)
```

getSIPPortRange(names) throws ObjectDoesNotExist

Get the port range used for SIP data connections, for each of the named virtual servers. This setting is only used when the SIP virtual server is using 'Full Gateway' mode.

```
VirtualServer.PortRange[] getSIPPortRange(
   String[] names
)
```

getSIPStreamingTimeout(names) throws ObjectDoesNotExist

Get the time, in seconds, after which a UDP stream will timeout if it has not seen any data.

```
Unsigned Integer[] getSIPStreamingTimeout(
   String[] names
)
```

getSSLCertificate(names) throws ObjectDoesNotExist

Get the name of the default SSL Certificate that is used for SSL decryption for each of the named virtual servers. This is the name of an item in the SSL Certificates Catalog.

```
String[] getSSLCertificate(
   String[] names
)
```

getSSLClientCertificateAuthorities(names) throws ObjectDoesNotExist

Get the certificate authorities that are trusted for validating client certificates, for each of the named virtual servers.

```
String[][] getSSLClientCertificateAuthorities(
   String[] names
)
```

getSSLClientCertificateHeaders(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should add HTTP headers to each request to show the data in the client certificate.

```
VirtualServer.SSLClientCertificateHeaders[]
getSSLClientCertificateHeaders(
   String[] names
)
```

getSSLDecrypt(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should decrypt SSL traffic.

```
Boolean[] getSSLDecrypt(
   String[] names
)
```

getSSLExpectStartTLS(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should upgrade SMTP connections to SSL using the STARTTLS command.

```
Boolean[] getSSLExpectStartTLS(
   String[] names
)
```

getSSLHeaders(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should add HTTP headers to each request to show SSL connection parameters.

```
Boolean[] getSSLHeaders(
   String[] names
)
```

getSSLLogEnabled(names) throws ObjectDoesNotExist

This method is now obsolete. SSL logging is now done if LogConnectionFailures is enabled. Use VirtualServer.getLogConnectionFailures and VirtualServer.getLogConnection failures to control this configuration.

```
Boolean[] getSSLLogEnabled(
   String[] names
)
```

getSSLRequestClientCertMode(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should request (or require) an identifying certificate from each client.

```
VirtualServer.SSLRequestClientCertMode[] getSSLRequestClientCertMode(
   String[] names
)
```

getSSLSendCloseAlerts(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should send a close alert when initiating SSL socket disconnections.

```
Boolean[] getSSLSendCloseAlerts(
   String[] names
)
```

getSSLSites(names) throws ObjectDoesNotExist

Gets a list of mappings between destination addresses and the certificate used for SSL decryption those addresses, for each of the named virtual servers. Each certificate is the name of an item in the SSL Certificates Catalog.

```
VirtualServer.SSLSite[][] getSSLSites(
   String[] names
)
```

getSSLTrustMagic(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should decode extra information on the true origin of an SSL connection. This information is prefixed onto an incoming SSL connection from another traffic manager.

```
Boolean[] getSSLTrustMagic(
   String[] names
)
```

getServerfirstBanner(names) throws ObjectDoesNotExist

Get the banner that each of the named virtual servers sends to clients for server-first protocols such as POP, SMTP and IMAP.

```
String[] getServerfirstBanner(
   String[] names
)
```

getServiceLevelMonitoring(names) throws ObjectDoesNotExist

Get the Service Level Monitoring class that each of the named virtual servers uses.

```
String[] getServiceLevelMonitoring(
   String[] names
)
```

getSipTransactionTimeout(names) throws ObjectDoesNotExist

Get the time after which an incomplete transaction should be discarded, in seconds, for each of the named virtual servers.

```
Unsigned Integer[] getSipTransactionTimeout(
   String[] names
)
```

getTimeout(names) throws ObjectDoesNotExist

Get the time to wait for data on an already established connection, in seconds, for each of the named virtual servers.

```
Unsigned Integer[] getTimeout(
   String[] names
)
```

getUDPResponseDatagramsExpected(names) throws ObjectDoesNotExist

Get the expected number of UDP datagrams in the response, for each of the named virtual servers. For simple request/response protocols a value of '1' should be used. If set to -1, the connection will not be discarded until the udp_timeout is reached.

```
Integer[] getUDPResponseDatagramsExpected(
   String[] names
)
```

getUDPTimeout(names) throws ObjectDoesNotExist

Get the time after which an idle UDP connection should be discarded and resources reclaimed, in seconds, for each of the named virtual servers.

```
Unsigned Integer[] getUDPTimeout(
   String[] names
)
```

getUseNagle(names) throws ObjectDoesNotExist

Get whether Nagle's algorithm should be used for TCP connections.

```
Boolean[] getUseNagle(
   String[] names
)
```

getVirtualServerNames()

Gets the names of all the configured virtual servers.

```
String[] getVirtualServerNames()
```

getWebcacheControlOut(names) throws ObjectDoesNotExist

Get the Cache-Control header that should be sent with cached HTTP responses.

```
String[] getWebcacheControlOut(
   String[] names
)
```

getWebcacheEnabled(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should attempt to cache web server responses.

```
Boolean[] getWebcacheEnabled(
   String[] names
)
```

getWebcacheErrorpageTime(names) throws ObjectDoesNotExist

Get the time periods that each of the named virtual servers should cache error pages for.

```
Unsigned Integer[] getWebcacheErrorpageTime(
   String[] names
)
```

getWebcacheRefreshTime(names) throws ObjectDoesNotExist

Get the time periods that each of the named virtual servers should consider re-fetching cached pages in.

```
Unsigned Integer[] getWebcacheRefreshTime(
   String[] names
)
```

getWebcacheTime(names) throws ObjectDoesNotExist

Get the time periods that each of the named virtual servers should cache web pages for.

```
Unsigned Integer[] getWebcacheTime(
   String[] names
)
```

removeCompressionMIMETypes(names, values) throws ObjectDoesNotExist, DeploymentError

For each named virtual server, remove new MIME types from the list of types to compress.

```
void removeCompressionMIMETypes(
   String[] names
   String[][] values
)
```

removeFTPPortRange(names) throws ObjectDoesNotExist, DeploymentError

Allow FTP connections to use any free ports, for each of the named virtual servers.

```
void removeFTPPortRange(
   String[] names
)
```

removeRTSPPortRange(names) throws ObjectDoesNotExist, DeploymentError

Allow any free ports to be used for RTSP connections, for each of the named virtual servers.

```
void removeRTSPPortRange(
   String[] names
)
```

removeResponseRules(names, rules) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, remove rules from the list that are run on server responses.

```
void removeResponseRules(
   String[] names
   String[][] rules
)
```

removeRules(names, rules) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, remove rules from the list of rules that are run on client requests.

```
void removeRules(
   String[] names
   String[][] rules
)
```

removeSIPPortRange(names) throws ObjectDoesNotExist, DeploymentError

Allow any free ports to be used for SIP connections, for each of the named virtual servers. This setting is only used when the SIP virtual server is using 'Full Gateway' mode.

```
void removeSIPPortRange(
   String[] names
)
```

removeSSLClientCertificateAuthorities(names, values) throws ObjectDoesNotExist, DeploymentError

Remove certificate authorities for validating client certificates for each of the named virtual servers.

```
void removeSSLClientCertificateAuthorities(
   String[] names
   String[][] values
)
```

renameVirtualServer(names, new_names) throws ObjectDoesNotExist, InvalidObjectName, DeploymentError, InvalidOperation

Rename each of the named virtual servers.

```
void renameVirtualServer(
   String[] names
   String[] new_names
)
```

setAddClusterClientIPHeader(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether a 'X-Cluster-Client-Ip' header should be added to each HTTP request, for each of the named virtual servers. The 'X-Cluster-Client-Ip' header contains the client's IP address.

```
void setAddClusterClientIPHeader(
   String[] names
   Boolean[] values
)
```

setApplicationFirewallEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation

For each of the named virtual servers, enable or disable the Zeus Application Firewall Module.

```
void setApplicationFirewallEnabled(
   String[] names
   Boolean[] values
)
```

setBandwidthClass(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Bandwidth Class that each of the named virtual servers uses.

```
void setBandwidthClass(
   String[] names
   String[] values
)
```

setCompressUnknownSize(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should compress documents with no given size.

```
void setCompressUnknownSize(
   String[] names
   Boolean[] values
)
```

setCompressionEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should compress web pages before sending to the client.

```
void setCompressionEnabled(
   String[] names
   Boolean[] values
)
```

setCompressionLevel(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the gzip compression level, for each of the named virtual servers.

```
void setCompressionLevel(
```

```
   String[] names
   Unsigned Integer[] values
)
```

setCompressionMIMETypes(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the list of MIME types to compress, for each of the named virtual servers.

```
void setCompressionMIMETypes(
   String[] names
   String[][] values
)
```

setCompressionMaxSize(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the maximum document size to compress, in bytes, for each of the named virtual servers. A document size of '0' means 'unlimited'.

```
void setCompressionMaxSize(
   String[] names
   Unsigned Integer[] values
)
```

setCompressionMinSize(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the minimum document size to compress, in bytes, for each of the named virtual servers.

```
void setCompressionMinSize(
   String[] names
   Unsigned Integer[] values
)
```

setConnectTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time to wait for data from a new connection, in seconds, for each of the named virtual servers. If no data is received in this time, the connection will be closed.

```
void setConnectTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setCookieDomainRewriteMode(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set how each of the named virtual servers should rewrite the domain portion of cookies set by a back-end web server.

```
void setCookieDomainRewriteMode(
   String[] names
   VirtualServer.CookieDomainRewriteMode[] values
)
```

setCookieNamedDomain(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the domain to use when rewriting cookie domains, for each of the named virtual servers.

```
void setCookieNamedDomain(
   String[] names
   String[] values
)
```

setCookiePathRewrite(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, set the regex, and replacement for rewriting the path portion of a cookie.

```
void setCookiePathRewrite(
   String[] names
   VirtualServer.RegexReplacement[] values
)
```

setCookieSecureFlagRewriteMode(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should modify the 'secure' tag of cookies set by a back-end web server.

```
void setCookieSecureFlagRewriteMode(
   String[] names
   VirtualServer.CookieSecureFlagRewriteMode[] values
)
```

setDefaultPool(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the default Pool that traffic is sent to for each of the named virtual servers.

```
void setDefaultPool(
   String[] names
   String[] values
)
```

setEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers is enabled (i.e. serving traffic).

```
void setEnabled(
   String[] names
   Boolean[] values
)
```

setErrorFile(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the file names of the error texts that each of the named virtual servers will send back to a client in case of back-end or internal errors.

```
void setErrorFile(
   String[] names
   String[] values
)
```

setFTPDataSourcePort(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the source port each of the named virtual servers should use for active-mode FTP data connections. If 0, a random high port will be used, otherwise the specified port will be used. If a port below 1024 is required you must first explicitly permit use of low ports with the ftp_data_bind_low global setting.

```
void setFTPDataSourcePort(
   String[] names
   Unsigned Integer[] values
)
```

setFTPForceClientSecure(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should require incoming FTP data connections (from clients) to originate from the same IP address as the corresponding control connection.

```
void setFTPForceClientSecure(
   String[] names
```

```
    Boolean[] values
)
```

setFTPForceServerSecure(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should require incoming FTP data connections (from nodes) to originate from the same IP address as the corresponding control connection.

```
void setFTPForceServerSecure(
    String[] names
    Boolean[] values
)
```

setFTPPortRange(names, range) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port range used for FTP data connections for each of the named virtual servers.

```
void setFTPPortRange(
    String[] names
    VirtualServer.FTPPortRange[] range
)
```

setFTPSSLData(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should use SSL on the data connection as well as the control connection

```
void setFTPSSLData(
    String[] names
    Boolean[] values
)
```

setKeepalive(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should allow clients to maintain keepalive connections.

```
void setKeepalive(
    String[] names
    Boolean[] values
)
```

setKeepaliveTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time that an idle keepalive connection should be kept open for, in seconds, for each of the named virtual servers.

```
void setKeepaliveTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setListenAddresses(names, addresses) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the specific IP addresses and hostnames for each named virtual server to listen on.

```
void setListenAddresses(
   String[] names
   String[][] addresses
)
```

setListenOnAllAddresses(names) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Make each of the named virtual servers listen on all IP addresses.

```
void setListenOnAllAddresses(
   String[] names
)
```

setListenTrafficIPGroups(names, groups) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, set specific Traffic IP Groups for it to listen on.

```
void setListenTrafficIPGroups(
   String[] names
   String[][] groups
)
```

setLocationDefaultRewriteMode(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should rewrite the 'Location' header. The rewrite is only performed if the location rewrite regex didn't match.

```
void setLocationDefaultRewriteMode(
   String[] names
   VirtualServer.LocationDefaultRewriteMode[] values
```

```
)
```

setLocationRewrite(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, set the regex, and replacement for rewriting any 'Location' headers.

```
void setLocationRewrite(
   String[] names
   VirtualServer.RegexReplacement[] values
)
```

setLogClientConnectionFailures(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log client connection failures.

```
void setLogClientConnectionFailures(
   String[] names
   Boolean[] values
)
```

setLogEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should log each connection to a disk on the file system.

```
void setLogEnabled(
   String[] names
   Boolean[] values
)
```

setLogFilename(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the name of the file used to store request logs, for each of the named virtual servers.

```
void setLogFilename(
   String[] names
   String[] values
)
```

setLogFormat(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the log file format for each of the named virtual servers.

```
void setLogFormat(
   String[] names
   String[] values
)
```

setLogServerConnectionFailures(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log server connection failures.

```
void setLogServerConnectionFailures(
   String[] names
   Boolean[] values
)
```

setMIMEAutoDetect(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should auto-detect MIME types if the server does not provide them.

```
void setMIMEAutoDetect(
   String[] names
   Boolean[] values
)
```

setMIMEDefaultType(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the MIME type that the server uses as its 'default', for each of the named virtual servers. Responses with this mime type will be auto-corrected by the virtual server if this setting is enabled.

```
void setMIMEDefaultType(
   String[] names
   String[] values
)
```

setMaxClientBuffer(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the amount of memory used to store data sent by the client, in bytes, for each of the named virtual servers.

```
void setMaxClientBuffer(
   String[] names
   Unsigned Integer[] values
)
```

setMaxServerBuffer(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the amount of memory used to store data returned by the server, in bytes, for each of the named virtual servers.

```
void setMaxServerBuffer(
   String[] names
   Unsigned Integer[] values
)
```

setNote(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the note for each of the named virtual servers.

```
void setNote(
   String[] names
   String[] values
)
```

setPort(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port that each of the named virtual servers listens on for incoming connections.

```
void setPort(
   String[] names
   Unsigned Integer[] values
)
```

setProtection(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Service Protection Settings that are used to protect each of the named virtual servers.

```
void setProtection(
   String[] names
   String[] values
)
```

setProtocol(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the protocol that each of the named virtual servers uses.

```
void setProtocol(
   String[] names
   VirtualServer.Protocol[] values
)
```

setProxyClose(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should send a FIN packet on to the back-end server when it is received from the client. The alternative is to close the connection to the client immediately. If your traffic manager is responding to the request itself, enabling this setting will cause your traffic manager to continue writing the response even after it has received a FIN from the client.

```
void setProxyClose(
   String[] names
   Boolean[] values
)
```

setRTSPPortRange(names, range) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port range used for RTSP streaming data connections for each of the named virtual servers.

```
void setRTSPPortRange(
   String[] names
   VirtualServer.PortRange[] range
)
```

setRequestSyslogEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should log each connection to a remote syslog server.

```
void setRequestSyslogEnabled(
   String[] names
   Boolean[] values
)
```

setRequestSyslogFormat(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the remote log line format for each of the named virtual servers.

```
void setRequestSyslogFormat(
   String[] names
   String[] values
)
```

setRequestSyslogIPEndpoint(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the remote syslog endpoint for each of the named virtual servers

```
void setRequestSyslogIPEndpoint(
   String[] names
   String[] values
)
```

setResponseRules(names, rules) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the rules that are run on server responses for each of the named virtual servers.

```
void setResponseRules(
   String[] names
   VirtualServer.Rule[][] rules
)
```

setRewriteSIPURI(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the Request-URI of SIP requests will be replaced with the selected back-end node's address.

```
void setRewriteSIPURI(
   String[] names
   Boolean[] values
)
```

setRules(names, rules) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the rules that are run on client requests for each of the named virtual servers.

```
void setRules(
   String[] names
   VirtualServer.Rule[][] rules
```

```
)
```

setSIPDangerousRequestMode(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set what should be done with requests that contain body data and should be routed to an external IP.

```
void setSIPDangerousRequestMode(
   String[] names
   VirtualServer.SIPDangerousRequestMode[] values
)
```

setSIPMaxConnectionMemory(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set maximum memory per connection.

```
void setSIPMaxConnectionMemory(
   String[] names
   Unsigned Integer[] values
)
```

setSIPMode(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set which mode of operation the SIP virtual server should run in.

```
void setSIPMode(
   String[] names
   VirtualServer.SIPMode[] values
)
```

setSIPPortRange(names, range) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port range used for SIP data connections for each of the named virtual servers. This setting is only used when the SIP virtual server is using 'Full Gateway' mode.

```
void setSIPPortRange(
   String[] names
   VirtualServer.PortRange[] range
)
```

setSIPStreamingTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time, in seconds, after which a UDP stream will timeout if it has not seen any data.

```
void setSIPStreamingTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setSSLCertificate(names, certs) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the name of the default SSL Certificate that is used for SSL decryption for each of the named virtual servers. This is the name of an item in the SSL Certificates Catalog. You must call this function to set an SSL Certificate before turning on SSL Decryption.

```
void setSSLCertificate(
   String[] names
   String[] certs
)
```

setSSLClientCertificateAuthorities(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the certificate authorities that are trusted for validating client certificates, for each of the named virtual servers.

```
void setSSLClientCertificateAuthorities(
   String[] names
   String[][] values
)
```

setSSLClientCertificateHeaders(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should add HTTP headers to each request to show the data in the client certificate.

```
void setSSLClientCertificateHeaders(
   String[] names
   VirtualServer.SSLClientCertificateHeaders[] values
)
```

setSSLDecrypt(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets whether each of the named virtual servers should decrypt SSL traffic. This function will error unless an SSL Certificate has previously been set using setSSLCertificate.

```
void setSSLDecrypt(
   String[] names
   Boolean[] values
)
```

setSSLExpectStartTLS(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should upgrade SMTP connections to SSL using the STARTTLS command.

```
void setSSLExpectStartTLS(
   String[] names
   Boolean[] values
)
```

setSSLHeaders(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should add HTTP headers to each request to show SSL connection parameters.

```
void setSSLHeaders(
   String[] names
   Boolean[] values
)
```

setSSLLogEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

This method is now obsolete. SSL logging is now done if LogConnectionFailures is enabled. Use VirtualServer.getLogConnectionFailures and VirtualServer.getLogConnection failures to control this configuration.

```
void setSSLLogEnabled(
   String[] names
   Boolean[] values
)
```

setSSLRequestClientCertMode(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should request (or require) an identifying certificate from each client.

```
void setSSLRequestClientCertMode(
   String[] names
   VirtualServer.SSLRequestClientCertMode[] values
)
```

setSSLSendCloseAlerts(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should send a close alert when initiating SSL socket disconnections.

```
void setSSLSendCloseAlerts(
   String[] names
   Boolean[] values
)
```

setSSLTrustMagic(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should decode extra information on the true origin of an SSL connection. This information is prefixed onto an incoming SSL connection from another traffic manager.

```
void setSSLTrustMagic(
   String[] names
   Boolean[] values
)
```

setServerfirstBanner(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the banner that each of the named virtual servers sends to clients for server-first protocols such as POP, SMTP and IMAP.

```
void setServerfirstBanner(
   String[] names
   String[] values
)
```

setServiceLevelMonitoring(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Service Level Monitoring class that each of the named virtual servers uses.

```
void setServiceLevelMonitoring(
   String[] names
   String[] values
)
```

setSipTransactionTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time after which an incomplete transaction should be discarded, in seconds, for each of the named virtual servers.

```
void setSipTransactionTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time to wait for data on an already established connection, in seconds, for each of the named virtual servers.

```
void setTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setUDPResponseDatagramsExpected(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the expected number of UDP datagrams in the response, for each of the named virtual servers. For simple request/response protocols a value of '1' should be used. If set to -1, the connection will not be discarded until the udp_timeout is reached.

```
void setUDPResponseDatagramsExpected(
   String[] names
   Integer[] values
)
```

setUDPTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time after which an idle UDP connection should be discarded and resources reclaimed, in seconds, for each of the named virtual servers.

```
void setUDPTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setUseNagle(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether Nagle's algorithm should be used for TCP connections.

```
void setUseNagle(
   String[] names
   Boolean[] values
)
```

setWebcacheControlOut(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Cache-Control header that should be sent with cached HTTP responses.

```
void setWebcacheControlOut(
   String[] names
   String[] values
)
```

setWebcacheEnabled(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should attempt to cache web server responses.

```
void setWebcacheEnabled(
   String[] names
   Boolean[] values
)
```

setWebcacheErrorpageTime(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time periods that each of the named virtual servers should cache error pages for.

```
void setWebcacheErrorpageTime(
   String[] names
```

```
   Unsigned Integer[] values
)
```

setWebcacheRefreshTime(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time periods that each of the named virtual servers should consider re-fetching cached pages in.

```
void setWebcacheRefreshTime(
   String[] names
   Unsigned Integer[] values
)
```

setWebcacheTime(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time periods that each of the named virtual servers should cache web pages for.

```
void setWebcacheTime(
   String[] names
   Unsigned Integer[] values
)
```

6.2.2 Structures

VirtualServer.BasicInfo

This structure contains the basic information for a virtual server. It is used when creating a server, or modifying the port, protocol or default pool of a server.

```
struct VirtualServer.BasicInfo {
   # The port to listen for incoming connections on.
   Integer port;

   # The protocol that this virtual server handles.
   VirtualServer.Protocol protocol;

   # The default pool that traffic to this virtual server will go
   # to.
   String default_pool;
}
```

VirtualServer.FTPPortRange

This structure contains the range of ports that FTP data connections use.

```
struct VirtualServer.FTPPortRange {
   # The lower bound of the port range for FTP data connections.
```

```
    Integer low;

    # The upper bound of the port range for FTP data connections.
    Integer high;
}
```

VirtualServer.PortRange

This structure contains the range of ports.

```
struct VirtualServer.PortRange {
    # The lower bound of the port range.
    Integer low;

    # The upper bound of the port range.
    Integer high;
}
```

VirtualServer.RegexReplacement

This structure contains a regex and a replacement string.

```
struct VirtualServer.RegexReplacement {
    # The regular expression used to match against.
    String regex;

    # The replacement string if the regular expression matches.
    # Parameters $1-$9 can be used to represent bracketed parts of
    # the regular expression.
    String replacement;
}
```

VirtualServer.Rule

This structure contains the information on how a rule is assigned to a virtual server.

```
struct VirtualServer.Rule {
    # The name of the rule.
    String name;

    # Whether the rule is enabled or not.
    Boolean enabled;

    # Whether the rule runs on every request/response, or just the
    # first
    VirtualServer.RuleRunFlag run_frequency;
}
```

VirtualServer.SSLSite

This object represents a mapping between a destination address and an SSL certificate (this is the name of an item in the SSL Certificates Catalog). Clients connecting to the SSL Site's address will be sent the associated certificate.

```
struct VirtualServer.SSLSite {
   # The destination address that this site handles.
   String dest_address;

   # The certificate that will be sent when clients connect to
   # the destination address. This is a certificate name from the
   # SSL Certificates Catalog.
   String certificate;
}
```

6.2.3 Enumerations

VirtualServer.CookieDomainRewriteMode

```
enum VirtualServer.CookieDomainRewriteMode {
   # Do not rewrite the domain
   no_rewrite,

   # Rewrite the domain to the host header of the request
   set_to_request,

   # Rewrite the domain to the named domain value
   set_to_named
}
```

VirtualServer.CookieSecureFlagRewriteMode

```
enum VirtualServer.CookieSecureFlagRewriteMode {
   # Do not modify the 'secure' tag
   no_modify,

   # Set the 'secure' tag
   set_secure,

   # Unset the 'secure' tag
   unset_secure
}
```

VirtualServer.LocationDefaultRewriteMode

```
enum VirtualServer.LocationDefaultRewriteMode {
   # Nothing;
   never,
```

```
    # Rewrite the hostname to the request's 'Host' header, and
    # rewrite the protocol and port if necessary;
    always,

    # Do not rewrite the hostname. Rewrite the protocol and port
    # if the hostname matches the request's 'Host' header.
    if_host_matches
}
```

VirtualServer.Protocol

```
enum VirtualServer.Protocol {
    # HTTP
    http,

    # FTP
    ftp,

    # IMAPv2
    imapv2,

    # IMAPv3
    imapv3,

    # IMAPv4
    imapv4,

    # POP3
    pop3,

    # SMTP
    smtp,

    # LDAP
    ldap,

    # Telnet
    telnet,

    # SSL
    ssl,

    # SSL (HTTPS)
    https,

    # SSL (IMAPS)
    imaps,

    # SSL (POP3S)
    pop3s,
```

```
   # SSL (LDAPS)
   ldaps,

   # UDP - Streaming
   udpstreaming,

   # UDP
   udp,

   # DNS (UDP)
   dns,

   # DNS (TCP)
   dns_tcp,

   # SIP (UDP)
   sipudp,

   # SIP (TCP)
   siptcp,

   # RTSP
   rtsp,

   # Generic server first
   server_first,

   # Generic client first
   client_first
}
```

VirtualServer.RuleRunFlag

This enumeration defines the run flags for a particular rule.

```
enum VirtualServer.RuleRunFlag {
   # Run on every request or response.
   run_every,

   # Run only on the first request or response.
   only_first
}
```

VirtualServer.SIPDangerousRequestMode

```
enum VirtualServer.SIPDangerousRequestMode {
   # Send the request to a back-end node
   node,
```

```
   # Send a 403 Forbidden response to the client
   forbid,

   # Forward the request to its target URI (dangerous)
   forward
}
```

VirtualServer.SIPMode

```
enum VirtualServer.SIPMode {
   # SIP Routing
   route,

   # SIP Gateway
   sipgw,

   # Full Gateway
   fullgw
}
```

VirtualServer.SSLClientCertificateHeaders

```
enum VirtualServer.SSLClientCertificateHeaders {
   # No data
   none,

   # Certificate fields
   simple,

   # Certificate fields and certificate text
   all
}
```

VirtualServer.SSLRequestClientCertMode

```
enum VirtualServer.SSLRequestClientCertMode {
   # Do not request a client certificate
   dont_request,

   # Request, but do not require a client certificate
   request,

   # Require a client certificate
   require
}
```

6.3 Pool

URI: `http://soap.zeus.com/zxtm/1.0/Pool/`

The Pool interface allows management of Pool objects. Using this interface, you can create, delete and rename pool objects, and manage their configuration.

6.3.1 Methods

addDrainingNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add nodes to the lists of draining nodes, for each of the named pools.

```
void addDrainingNodes(
   String[] names
   String[][] values
)
```

addMonitors(names, values) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

Add monitors to each of the named pools.

```
void addMonitors(
   String[] names
   String[][] values
)
```

addNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add nodes to each of the named pools.

```
void addNodes(
   String[] names
   String[][] values
)
```

addPool(names, nodes) throws ObjectAlreadyExists, InvalidObjectName, InvalidInput, DeploymentError

Add each of the named pools, using the node lists for each.

```
void addPool(
   String[] names
   String[][] nodes
)
```

copyPool(names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError

Copy each of the named pools.

```
void copyPool(
   String[] names
   String[] new_names
)
```

deletePool(names) throws ObjectInUse, ObjectDoesNotExist, DeploymentError

Delete each of the named pools.

```
void deletePool(
   String[] names
)
```

disableNodes(names, nodes) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

For each of the named pools, disable the specified nodes in the pool.

```
void disableNodes(
   String[] names
   String[][] nodes
)
```

enableNodes(names, nodes) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

For each of the named pools, enable the specified nodes that are disabled in the pool.

```
void enableNodes(
   String[] names
   String[][] nodes
)
```

getBandwidthClass(names) throws ObjectDoesNotExist

Get the Bandwidth Classes that each of the named pools uses.

```
String[] getBandwidthClass(
   String[] names
)
```

getDisabledNodes(names) throws ObjectDoesNotExist

For each of the named pools, get the disabled nodes in the pool.

```
String[][] getDisabledNodes(
   String[] names
)
```

getDrainingNodes(names) throws ObjectDoesNotExist

Get the lists of draining nodes for each of the named pools.

```
String[][] getDrainingNodes(
   String[] names
)
```

getErrorFile(names) throws ObjectDoesNotExist

This method is now obsolete and is replaced by VirtualServer.getErrorFile.

```
String[] getErrorFile(
   String[] names
)
```

getFTPSupportRfc2428(names) throws ObjectDoesNotExist

Get whether backend IPv4 nodes understand the FTP EPRT and EPSV commands.

```
Boolean[] getFTPSupportRfc2428(
   String[] names
)
```

getFailpool(names) throws ObjectDoesNotExist

Get the pool to use when all nodes in a pool fail, for each of the named pools.

```
String[] getFailpool(
   String[] names
)
```

getKeepalive(names) throws ObjectDoesNotExist

Get whether each of the named pools should maintain HTTP keepalive connections to the nodes.

```
Boolean[] getKeepalive(
   String[] names
)
```

getLoadBalancingAlgorithm(names) throws ObjectDoesNotExist

Get the load balancing algorithms that each of the named pools uses.

```
Pool.LoadBalancingAlgorithm[] getLoadBalancingAlgorithm(
   String[] names
)
```

getMaxConnectTime(names) throws ObjectDoesNotExist

Get the times that each of the named pools should wait for a connection to establish to a node before trying another node, in seconds.

```
Unsigned Integer[] getMaxConnectTime(
   String[] names
)
```

getMaxIdleConnectionsPerNode(names) throws ObjectDoesNotExist

Get the maximum numbers of unused HTTP keepalive connections that each of the named pools should maintain to an individual node.

```
Unsigned Integer[] getMaxIdleConnectionsPerNode(
   String[] names
)
```

getMaxKeepalivesPerNode(names) throws ObjectDoesNotExist

getMaxKeepalivesPerNode is deprecated, please use getMaxIdleConnectionsPerNode instead.

```
Unsigned Integer[] getMaxKeepalivesPerNode(
   String[] names
)
```

getMaxReplyTime(names) throws ObjectDoesNotExist

Get the time that each of the named pools should wait for a response from a node before either discarding the request or trying another node, in seconds (retryable requests only).

```
Unsigned Integer[] getMaxReplyTime(
   String[] names
)
```

getMonitors(names) throws ObjectDoesNotExist

Get the list of all monitors.

```
String[][] getMonitors(
```

```
   String[] names
)
```

getNodeConnClose(names) throws ObjectDoesNotExist

Get whether all connections that have been sent to a node are closed when that node is marked as dead.

```
Boolean[] getNodeConnClose(
   String[] names
)
```

getNodeConnectionAttempts(names) throws ObjectDoesNotExist

Get the number of times your traffic manager should try and connect to a node before registering it as failed.

```
Unsigned Integer[] getNodeConnectionAttempts(
   String[] names
)
```

getNodeFailTime(names) throws ObjectDoesNotExist

Get the length of time a failed node should be isolated for before testing it with new traffic, in seconds

```
Unsigned Integer[] getNodeFailTime(
   String[] names
)
```

getNodes(names) throws ObjectDoesNotExist

Get the lists of nodes for each of the named pools.

```
String[][] getNodes(
   String[] names
)
```

getNodesConnectionCounts(nodes)

Get the number of active connections to each of the specified nodes.

```
Integer[] getNodesConnectionCounts(
   String[] nodes
)
```

getNodesLastUsed(nodes)

Get the number of seconds since each of the specified nodes was last used.

```
Integer[] getNodesLastUsed(
   String[] nodes
)
```

getNodesPriorityValue(names, nodes) throws ObjectDoesNotExist

For each of the named pools, get the priority values for the named nodes in each pool.

```
Pool.PriorityValueDefinition[][] getNodesPriorityValue(
   String[] names
   String[][] nodes
)
```

getNodesWeightings(names, nodes) throws ObjectDoesNotExist

For each of the named pools, get the weighting values for the specified nodes in this pool.

```
Pool.WeightingsDefinition[][] getNodesWeightings(
   String[] names
   String[][] nodes
)
```

getNote(names) throws ObjectDoesNotExist

Get the note for each of the named pools.

```
String[] getNote(
   String[] names
)
```

getPassiveMonitoring(names) throws ObjectDoesNotExist

Get whether this pool uses passive monitoring.

```
Boolean[] getPassiveMonitoring(
   String[] names
)
```

getPersistence(names) throws ObjectDoesNotExist

Get the default Session Persistence classes that each of the named pools uses.

```
String[] getPersistence(
   String[] names
)
```

getPoolNames()

Get the names of all of the configured pools.

```
String[] getPoolNames()
```

getPriorityEnabled(names) throws ObjectDoesNotExist

Get whether each of the named pools uses priority lists.

```
Boolean[] getPriorityEnabled(
   String[] names
)
```

getPriorityNodes(names) throws ObjectDoesNotExist

Get the minimum number of highest-priority active nodes, for each of the named pools.

```
Unsigned Integer[] getPriorityNodes(
   String[] names
)
```

getPriorityValues(names) throws ObjectDoesNotExist

For each of the named pools, get the priority values for each of the nodes in each pool.

```
Pool.PriorityValueDefinition[][] getPriorityValues(
   String[] names
)
```

getSMTPSendStartTLS(names) throws ObjectDoesNotExist

Get whether each of the named pools should upgrade SMTP connections to SSL using STARTTLS (the alternative is to encrypt the entire connection).

```
Boolean[] getSMTPSendStartTLS(
   String[] names
)
```

getSSLClientAuth(names) throws ObjectDoesNotExist

Get whether each of the named pools should use client authentication. If client authentication is enabled and a back-end node asks for a client authentication, a suitable certificate and private key will be used from the SSL Client Certificates catalog.

```
Boolean[] getSSLClientAuth(
   String[] names
)
```

getSSLEncrypt(names) throws ObjectDoesNotExist

Get whether each of the named pools should encrypt data to the back-end nodes using SSL.

```
Boolean[] getSSLEncrypt(
   String[] names
)
```

getSSLEnhance(names) throws ObjectDoesNotExist

Get whether each of the named pools should use SSL protocol enhancements. These enhancements allow Zeus Web Servers to run multiple SSL sites, and to discover the client's IP address. Only use enable this if, for this pool, you are using Zeus Web Servers or Zeus Traffic Managers whose virtual servers have the 'ssl_trust_magic' setting enabled.

```
Boolean[] getSSLEnhance(
   String[] names
)
```

getSSLSendCloseAlerts(names) throws ObjectDoesNotExist

Get whether each of the named pools should send a close alert when they initiate socket disconnections.

```
Boolean[] getSSLSendCloseAlerts(
   String[] names
)
```

getSSLServerNameExtension(names) throws ObjectDoesNotExist

Get if we should send the server_name extension to the back-end node. This setting forces the use of at least TLS 1.0.

```
Boolean[] getSSLServerNameExtension(
   String[] names
)
```

getSSLStrictVerify(names) throws ObjectDoesNotExist

Get whether each of the named pools should perform strict certificate validation on SSL certificates from the back-end nodes.

```
Boolean[] getSSLStrictVerify(
   String[] names
)
```

getTransparent(names) throws ObjectDoesNotExist

Get whether each of the named pools should make connections to the back-ends appear to originate from the source client IP address.

```
Boolean[] getTransparent(
   String[] names
)
```

getWeightings(names) throws ObjectDoesNotExist

For each of the named pools, get the weightings for each of the nodes in each pool.

```
Pool.WeightingsDefinition[][] getWeightings(
   String[] names
)
```

removeDrainingNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove nodes from the lists of draining nodes, for each of the named pools.

```
void removeDrainingNodes(
   String[] names
   String[][] values
)
```

removeMonitors(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove monitors from each of the named pools.

```
void removeMonitors(
   String[] names
   String[][] values
)
```

removeNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove nodes from each of the named pools.

```
void removeNodes(
   String[] names
   String[][] values
)
```

renamePool(names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename each of the named pools.

```
void renamePool(
   String[] names
   String[] new_names
)
```

setBandwidthClass(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the Bandwidth Classes that each of the named pools uses.

```
void setBandwidthClass(
   String[] names
   String[] values
)
```

setDisabledNodes(names, nodes) throws ObjectDoesNotExist, InvalidInput, DeploymentError

For each of the named pools, set the specified nodes to be disabled in the pool (all other nodes will be enabled).

```
void setDisabledNodes(
   String[] names
   String[][] nodes
)
```

setDrainingNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the lists of draining nodes for each of the named pools.

```
void setDrainingNodes(
   String[] names
   String[][] values
)
```

setErrorFile(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

This method is now obsolete and is replaced by VirtualServer.setErrorFile.

```
void setErrorFile(
   String[] names
   String[] values
```

```
)
```

setFTPSupportRfc2428(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether backend IPv4 nodes understand the FTP EPRT and EPSV commands.

```
void setFTPSupportRfc2428(
   String[] names
   Boolean[] values
)
```

setFailpool(names, values) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

Set the pool to use when all nodes in a pool fail, for each of the named pools.

```
void setFailpool(
   String[] names
   String[] values
)
```

setKeepalive(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should maintain HTTP keepalive connections to the nodes.

```
void setKeepalive(
   String[] names
   Boolean[] values
)
```

setLoadBalancingAlgorithm(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the load balancing algorithms that each of the named pools uses.

```
void setLoadBalancingAlgorithm(
   String[] names
   Pool.LoadBalancingAlgorithm[] values
)
```

setMaxConnectTime(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the times that each of the named pools should wait for a connection to establish to a node before trying another node, in seconds.

```
void setMaxConnectTime(
   String[] names
   Unsigned Integer[] values
)
```

setMaxIdleConnectionsPerNode(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum numbers of unused HTTP keepalive connections that each of the named pools should maintain to an individual node.

```
void setMaxIdleConnectionsPerNode(
   String[] names
   Unsigned Integer[] values
)
```

setMaxKeepalivesPerNode(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

setMaxKeepalivesPerNode is deprecated, please use setMaxIdleConnectionsPerNode instead.

```
void setMaxKeepalivesPerNode(
   String[] names
   Unsigned Integer[] values
)
```

setMaxReplyTime(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the time that each of the named pools should wait for a response from a node before either discarding the request or trying another node, in seconds (retryable requests only).

```
void setMaxReplyTime(
   String[] names
   Unsigned Integer[] values
)
```

setMonitors(names, values) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

Set the list of all monitors.

```
void setMonitors(
   String[] names
   String[][] values
)
```

setNodeConnClose(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether all connections that have been sent to a node are closed when that node is marked as dead.

```
void setNodeConnClose(
   String[] names
   Boolean[] values
)
```

setNodeConnectionAttempts(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the number of times your traffic manager should try and connect to a node before registering it as failed.

```
void setNodeConnectionAttempts(
   String[] names
   Unsigned Integer[] values
)
```

setNodeFailTime(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the length of time a failed node should be isolated for before testing it with new traffic, in seconds

```
void setNodeFailTime(
   String[] names
   Unsigned Integer[] values
)
```

setNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the lists of nodes for each of the named pools.

```
void setNodes(
   String[] names
   String[][] values
)
```

setNodesPriorityValue(names, node_values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

For each of the named pools, set the priority values for the named nodes in each pool.

```
void setNodesPriorityValue(
   String[] names
   Pool.PriorityValueDefinition[][] node_values
)
```

setNodesWeightings(names, nodes_values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

For each of the named pools, set the weighting (for the Weighted Round Robin algorithm) for each node in that pool.

```
void setNodesWeightings(
   String[] names
   Pool.WeightingsDefinition[][] nodes_values
)
```

setNote(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the note for each of the named pools.

```
void setNote(
   String[] names
   String[] values
)
```

setPassiveMonitoring(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether this pool uses passive monitoring.

```
void setPassiveMonitoring(
   String[] names
   Boolean[] values
)
```

setPersistence(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the default Session Persistence classes that each of the named pools uses.

```
void setPersistence(
   String[] names
   String[] values
```

```
)
```

setPriorityEnabled(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools uses priority lists.

```
void setPriorityEnabled(
   String[] names
   Boolean[] values
)
```

setPriorityNodes(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the minimum number of highest-priority active nodes, for each of the named pools.

```
void setPriorityNodes(
   String[] names
   Unsigned Integer[] values
)
```

setSMTPSendStartTLS(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should upgrade SMTP connections to SSL using STARTTLS (the alternative is to encrypt the entire connection).

```
void setSMTPSendStartTLS(
   String[] names
   Boolean[] values
)
```

setSSLClientAuth(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should use client authentication. If client authentication is enabled and a back-end node asks for a client authentication, a suitable certificate and private key will be used from the SSL Client Certificates catalog.

```
void setSSLClientAuth(
   String[] names
   Boolean[] values
)
```

setSSLEncrypt(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should encrypt data to the back-end nodes using SSL.

```
void setSSLEncrypt(
   String[] names
   Boolean[] values
)
```

setSSLEnhance(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should use SSL protocol enhancements. These enhancements allow Zeus Web Servers to run multiple SSL sites, and to discover the client's IP address. Only use enable this if, for this pool, you are using Zeus Web Servers or Zeus Traffic Managers whose virtual servers have the 'ssl_trust_magic' setting enabled.

```
void setSSLEnhance(
   String[] names
   Boolean[] values
)
```

setSSLSendCloseAlerts(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should send a close alert when they initiate socket disconnections.

```
void setSSLSendCloseAlerts(
   String[] names
   Boolean[] values
)
```

setSSLServerNameExtension(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set if we should send the server_name extension to the back-end node. This setting forces the use of at least TLS 1.0.

```
void setSSLServerNameExtension(
   String[] names
   Boolean[] values
)
```

setSSLStrictVerify(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should perform strict certificate validation on SSL certificates from the back-end nodes.

```
void setSSLStrictVerify(
   String[] names
   Boolean[] values
)
```

setTransparent(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should make connections to the back-ends appear to originate from the source client IP address.

```
void setTransparent(
   String[] names
   Boolean[] values
)
```

6.3.2 Structures

Pool.PriorityValueDefinition

This structure contains the priority for a particular node. The priority is used when using the priority lists functionality.

```
struct Pool.PriorityValueDefinition {
   # The name of the node.
   String node;

   # The priority value.
   Integer priority;
}
```

Pool.WeightingsDefinition

This structure contains the weighting for a particular node. The weighting is used when using the Weighted Round Robin algorithm functionality.

```
struct Pool.WeightingsDefinition {
   # The name of the node.
   String node;

   # The weighting value.
   Integer weighting;
}
```

6.3.3 Enumerations

Pool.LoadBalancingAlgorithm

```
enum Pool.LoadBalancingAlgorithm {
   # Round Robin
   roundrobin,

   # Weighted Round Robin
   wroundrobin,

   # Perceptive
   cells,

   # Least Connections
   connections,

   # Weighted Least Connections
   wconnections,

   # Fastest Response Time
   responsetimes,

   # Random node
   random
}
```

6.4 TrafficIPGroups

URI: `http://soap.zeus.com/zxtm/1.0/TrafficIPGroups/`

The TrafficIPGroup interface allows management of Traffic IP Group objects. Using this interface, you can create, delete and rename Traffic IP Group objects, and manage their configuration.

6.4.1 Methods

addIPAddresses(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Add new IP addresses to each of the named Traffic IP Groups.

```
void addIPAddresses(
   String[] names
   String[][] values
)
```

addPassiveMachine(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Add machines to the lists of passive machines, for each of the named Traffic IP Groups.

```
void addPassiveMachine(
   String[] names
   String[][] values
)
```

addTrafficIPGroup(names, details) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidInput

Add the new named Traffic IP Groups, using the provided details.

```
void addTrafficIPGroup(
   String[] names
   TrafficIPGroups.Details[] details
)
```

addTrafficManager(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add new Traffic Managers to each of the named Traffic IP Groups.

```
void addTrafficManager(
   String[] names
   String[][] values
)
```

deleteTrafficIPGroup(names) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, InvalidOperation

Delete the named Traffic IP Groups.

```
void deleteTrafficIPGroup(
   String[] names
)
```

getAvailableTrafficManagers()

Get the names of all of the Traffic Managers in the cluster.

```
String[] getAvailableTrafficManagers()
```

getEnabled(names) throws ObjectDoesNotExist

Get whether this Traffic IP Group is enabled or not.

```
Boolean[] getEnabled(
   String[] names
)
```

getIPAddresses(names) throws ObjectDoesNotExist

Get the IP addresses that are managed by each of the named Traffic IP Groups.

```
String[][] getIPAddresses(
   String[] names
)
```

getIPDistributionMode(names) throws ObjectDoesNotExist

Get how IP's will be distributed across the machines in the cluster. If 'multihosted' mode is used, the multicast IP must be set first.

```
TrafficIPGroups.IPDistributionMode[] getIPDistributionMode(
   String[] names
)
```

getKeepTogether(names) throws ObjectDoesNotExist

Get the KeepTogether attribute for each of the named Traffic IP Groups.

```
Boolean[] getKeepTogether(
   String[] names
)
```

getMulticastIP(names) throws ObjectDoesNotExist

Get the multicast IP group that is used to share data across machines in the cluster. This setting is only used if the Traffic IP is using 'shared' distribution mode.

```
String[] getMulticastIP(
   String[] names
)
```

getNote(names) throws ObjectDoesNotExist

Get the note for each of the named Traffic IP Groups.

```
String[] getNote(
   String[] names
)
```

getPassiveMachine(names) throws ObjectDoesNotExist

Get the lists of passive machines in each of the named Traffic IP Groups.

```
String[][] getPassiveMachine(
   String[] names
)
```

getTrafficIPGroupNames()

Get the names of all of the configured Traffic IP Groups.

```
String[] getTrafficIPGroupNames()
```

getTrafficManager(names) throws ObjectDoesNotExist

Get the Traffic Managers that manage the IP addresses in each of the named Traffic IP Groups.

```
String[][] getTrafficManager(
   String[] names
)
```

getUseClientSourcePort(names) throws ObjectDoesNotExist

Get whether the source port is taken into account when deciding which traffic manager should handle the request. This setting is only used if the Traffic IP is using 'multihosted' distribution mode.

```
Boolean[] getUseClientSourcePort(
   String[] names
)
```

removeIPAddresses(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Remove the named IP addresses from each of the named Traffic IP Groups.

```
void removeIPAddresses(
   String[] names
   String[][] values
)
```

removePassiveMachine(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Remove the named machines from the list of passive machines, for each of the named Traffic IP Groups.

```
void removePassiveMachine(
   String[] names
   String[][] values
)
```

removeTrafficManager(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidOperation

Remove the named Traffic Managers from each named Traffic IP Group.

```
void removeTrafficManager(
   String[] names
   String[][] values
)
```

renameTrafficIPGroup(names, new_names) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Rename each of the named Traffic IP Groups.

```
void renameTrafficIPGroup(
   String[] names
   String[] new_names
)
```

setEnabled(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether this Traffic IP Group is enabled or not.

```
void setEnabled(
   String[] names
   Boolean[] values
)
```

setIPAddresses(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the IP addresses that are managed by each of the named Traffic IP Groups.

```
void setIPAddresses(
   String[] names
   String[][] values
)
```

setIPDistributionMode(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set how IP's will be distributed across the machines in the cluster. If 'multihosted' mode is used, the multicast IP must be set first.

```
void setIPDistributionMode(
   String[] names
   TrafficIPGroups.IPDistributionMode[] values
)
```

setKeepTogether(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the KeepTogether attribute for each of the named Traffic IP Groups.

```
void setKeepTogether(
   String[] names
   Boolean[] values
)
```

setMulticastIP(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the multicast IP group that is used to share data across machines in the cluster. This setting is only used if the Traffic IP is using 'shared' distribution mode.

```
void setMulticastIP(
   String[] names
   String[] values
)
```

setNote(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the note for each of the named Traffic IP Groups.

```
void setNote(
   String[] names
   String[] values
)
```

setPassiveMachine(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the lists of passive machines in each of the named Traffic IP Groups.

```
void setPassiveMachine(
   String[] names
```

```
   String[][] values
)
```

setTrafficManager(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the Traffic Managers that manage the IP addresses in each of the named Traffic IP Groups.

```
void setTrafficManager(
   String[] names
   String[][] values
)
```

setUseClientSourcePort(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether the source port is taken into account when deciding which traffic manager should handle the request. This setting is only used if the Traffic IP is using 'multihosted' distribution mode.

```
void setUseClientSourcePort(
   String[] names
   Boolean[] values
)
```

6.4.2 Structures

TrafficIPGroups.Details

This structure contains the basic details of a Traffic IP Group: the nodes, and the traffic managers that the Traffic IP group spans. It is used when creating a new Traffic IP Group.

```
struct TrafficIPGroups.Details {
   # The IP addresses in the Traffic IP Group.
   String[] ipaddresses;

   # The names of the traffic managers that will manage the IP
   # Addresses.
   String[] machines;
}
```

6.4.3 Enumerations

TrafficIPGroups.IPDistributionMode

```
enum TrafficIPGroups.IPDistributionMode {
   # Raise each address on a single machine (Single-Hosted mode)
```

```
   singlehosted,

   # Raise each address on every machine in the group
   # (Multi-Hosted mode) - IPv4 only
   multihosted,

   # Use an EC2 Elastic IP address.
   ec2elastic
}
```

6.5 Catalog.Rule

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/Rule/`

The Catalog.Rule interface allows management of TrafficScript Rules. Using this interface, you can create, delete and rename rules, and manage their configuration. You can also syntax-check rule fragments.

6.5.1 Methods

addRule(names, texts) throws InvalidObjectName, ObjectAlreadyExists, DeploymentError

Add new rules to the catalog.

```
void addRule(
   String[] names
   String[] texts
)
```

checkSyntax(rule_text)

Check the syntax of each of the supplied TrafficScript rule texts. This method does not modify any configuration.

```
Catalog.Rule.SyntaxCheck[] checkSyntax(
   String[] rule_text
)
```

copyRule(names, new_names) throws InvalidObjectName, ObjectAlreadyExists, ObjectDoesNotExist, DeploymentError

Copy the named rules in the catalog.

```
void copyRule(
   String[] names
   String[] new_names
)
```

deleteRule(names) throws ObjectInUse, ObjectDoesNotExist, DeploymentError, InvalidOperation

Delete the named rules from the catalog.

```
void deleteRule(
   String[] names
)
```

getRuleDetails(names) throws ObjectDoesNotExist, DeploymentError

Get the rule text and notes (if any), for each of the named rules.

```
Catalog.Rule.RuleInfo[] getRuleDetails(
   String[] names
)
```

getRuleNames()

Get the names of all rules in the catalog.

```
String[] getRuleNames()
```

renameRule(names, new_names) throws InvalidObjectName, ObjectAlreadyExists, ObjectDoesNotExist, DeploymentError, InvalidOperation

Rename the named rules in the catalog.

```
void renameRule(
   String[] names
   String[] new_names
)
```

setRuleNotes(names, notes) throws ObjectDoesNotExist, DeploymentError

Sets the descriptive notes for each of the named rules in the catalog.

```
void setRuleNotes(
   String[] names
   String[] notes
)
```

setRuleText(names, text) throws ObjectDoesNotExist, DeploymentError

Set the TrafficScript text for each of the named rules in the catalog.

```
void setRuleText(
   String[] names
   String[] text
```

```
)
```

6.5.2 Structures

Catalog.Rule.RuleInfo

This structure contains basic information for a rule in the catalog.

```
struct Catalog.Rule.RuleInfo {
   # The rule text
   String rule_text;

   # The descriptive notes for the rule.
   String rule_notes;
}
```

Catalog.Rule.SyntaxCheck

This structure contains the results of a rule syntax check against a rule in the catalog.

```
struct Catalog.Rule.SyntaxCheck {
   # Whether the rule text is valid or not.
   Boolean valid;

   # Any warnings (such as deprecated functions) associated with
   # the rule text.
   String warnings;

   # Any errors (such as syntax errors) associated with the rule
   # text.
   String errors;
}
```

6.6 Catalog.Monitor

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/Monitor/`

The Catalog.Monitor interface allows management of Custom Monitors. Using this interface, you can create, delete and rename custom monitors, and manage their configuration.

6.6.1 Methods

addMonitors(names) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add new monitors (defaults to TCP transaction monitor, monitoring each node separately).

```
void addMonitors(
```

```
   String[] names
)
```

addProgramArguments(names, arguments) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Adds a set of arguments to the specified monitors. The monitors specified must be of type 'program'.

```
void addProgramArguments(
   String[] names
   Catalog.Monitor.Argument[][] arguments
)
```

copyMonitors(names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Copy the named monitors.

```
void copyMonitors(
   String[] names
   String[] new_names
)
```

deleteMonitorProgram(names) throws ObjectDoesNotExist, DeploymentError, ObjectInUse

Delete the named monitor programs.

```
void deleteMonitorProgram(
   String[] names
)
```

deleteMonitors(names) throws ObjectDoesNotExist, InvalidOperation, DeploymentError, ObjectInUse

Delete these monitors.

```
void deleteMonitors(
   String[] names
)
```

downloadMonitorProgram(name) throws ObjectDoesNotExist

Download the named monitor program.

```
Binary Data downloadMonitorProgram(
   String name
```

```
)
```

getAllMonitorNames()

Get the names of all monitors.

```
String[] getAllMonitorNames()
```

getAuthentication(names) throws ObjectDoesNotExist, InvalidOperation

Get the authentication (user:password) that each of the named monitors should use in the test HTTP request.

```
String[] getAuthentication(
   String[] names
)
```

getBodyRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the body regular expression that that each of the named monitors' HTTP response must match.

```
String[] getBodyRegex(
   String[] names
)
```

getCloseString(names) throws ObjectDoesNotExist, InvalidOperation

Get an optional string that each of the named monitors should write to the server before closing the connection.

```
String[] getCloseString(
   String[] names
)
```

getCustomMonitorNames()

Get the names of all the custom monitors.

```
String[] getCustomMonitorNames()
```

getDelay(names) throws ObjectDoesNotExist, InvalidOperation

Get the minimum time between calls to each of the named monitors (in seconds).

```
Unsigned Integer[] getDelay(
   String[] names
)
```

getFailures(names) throws ObjectDoesNotExist, InvalidOperation

Get the number of failures required, by each of the named monitors, before a node is classed as unavailable.

```
Unsigned Integer[] getFailures(
   String[] names
)
```

getHostHeader(names) throws ObjectDoesNotExist, InvalidOperation

Get the host header that each of the named monitors should use in the test HTTP request.

```
String[] getHostHeader(
   String[] names
)
```

getMachine(names) throws ObjectDoesNotExist, InvalidOperation

Get the machine that each of the named monitors should monitor (must be a valid hostname:port or a hostname for Ping monitors).

```
String[] getMachine(
   String[] names
)
```

getMaxResponseLen(names) throws ObjectDoesNotExist, InvalidOperation

Get the maximum amount of data (in bytes) that each of the named monitors should read back from a server (0 = unlimited).

```
Unsigned Integer[] getMaxResponseLen(
   String[] names
)
```

getMonitorProgramNames()

Get the names of all the uploaded monitor programs. These are the programs that can be executed by custom program monitors.

```
String[] getMonitorProgramNames()
```

getNote(names) throws ObjectDoesNotExist, InvalidOperation

Get the note for each of the named Monitors.

```
String[] getNote(
```

```
   String[] names
)
```

getPath(names) throws ObjectDoesNotExist, InvalidOperation

Get the path that each of the named monitors should use in the test HTTP request.

```
String[] getPath(
   String[] names
)
```

getProgram(names) throws ObjectDoesNotExist, InvalidOperation

Get the name of the program that each named monitor runs.

```
String[] getProgram(
   String[] names
)
```

getProgramArguments(names) throws ObjectDoesNotExist, InvalidOperation

Gets all arguments for the specified monitors. The monitors specified must be of type 'program'.

```
Catalog.Monitor.Argument[][] getProgramArguments(
   String[] names
)
```

getResponseRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the regular expression that each of the named monitors should match against the server response.

```
String[] getResponseRegex(
   String[] names
)
```

getRtspBodyRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the body regular expression that each of the named monitors' RTSP response must match.

```
String[] getRtspBodyRegex(
   String[] names
)
```

getRtspPath(names) throws ObjectDoesNotExist, InvalidOperation

Get the path that each of the named monitors should use in the test RTSP request.

```
String[] getRtspPath(
   String[] names
)
```

getRtspStatusRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the status code regular expression that each of the named monitors' RTSP response must match.

```
String[] getRtspStatusRegex(
   String[] names
)
```

getScope(names) throws ObjectDoesNotExist, InvalidOperation

Get the scope of each named monitor.

```
Catalog.Monitor.Scope[] getScope(
   String[] names
)
```

getSipBodyRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the body regular expression that that each of the named monitors' SIP response must match.

```
String[] getSipBodyRegex(
   String[] names
)
```

getSipStatusRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the status code regular expression that that each of the named monitors' SIP response must match.

```
String[] getSipStatusRegex(
   String[] names
)
```

getSipTransport(names) throws ObjectDoesNotExist, InvalidOperation

Get the transport protocol that the monitor will use

```
Catalog.Monitor.SipTransport[] getSipTransport(
   String[] names
```

```
)
```

getStatusRegex(names) throws ObjectDoesNotExist, InvalidOperation

Get the status code regular expression that that each of the named monitors' HTTP response must match.

```
String[] getStatusRegex(
   String[] names
)
```

getTimeout(names) throws ObjectDoesNotExist, InvalidOperation

Get the maximum time that an individual instance, of each of the named monitors, is allowed to run for (in seconds).

```
Unsigned Integer[] getTimeout(
   String[] names
)
```

getType(names) throws ObjectDoesNotExist, InvalidOperation

Get the internal monitor type to use for each named monitor.

```
Catalog.Monitor.Type[] getType(
   String[] names
)
```

getUseSSL(names) throws ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors can connect using SSL.

```
Boolean[] getUseSSL(
   String[] names
)
```

getVerbose(names) throws ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors should emit verbose logging (useful for diagnostics).

```
Boolean[] getVerbose(
   String[] names
)
```

getWriteString(names) throws ObjectDoesNotExist, InvalidOperation

Get the string that each of the named monitors should write down the TCP connection.

```
String[] getWriteString(
   String[] names
)
```

removeProgramArguments(names, arguments) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Removes a set of arguments from the specified monitors. The monitors specified must be of type 'program'.

```
void removeProgramArguments(
   String[] names
   String[][] arguments
)
```

renameMonitors(names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename these monitors.

```
void renameMonitors(
   String[] names
   String[] new_names
)
```

setAuthentication(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the authentication (user:password) that each of the named monitors should use in the test HTTP request.

```
void setAuthentication(
   String[] names
   String[] values
)
```

setBodyRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the body regular expression that that each of the named monitors' HTTP response must match.

```
void setBodyRegex(
   String[] names
   String[] values
)
```

setCloseString(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set an optional string that each of the named monitors should write to the server before closing the connection.

```
void setCloseString(
   String[] names
   String[] values
)
```

setDelay(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the minimum time between calls to each of the named monitors (in seconds).

```
void setDelay(
   String[] names
   Unsigned Integer[] values
)
```

setFailures(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the number of failures required, by each of the named monitors, before a node is classed as unavailable.

```
void setFailures(
   String[] names
   Unsigned Integer[] values
)
```

setHostHeader(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the host header that each of the named monitors should use in the test HTTP request.

```
void setHostHeader(
   String[] names
   String[] values
)
```

setMachine(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the machine that each of the named monitors should monitor (must be a valid hostname:port or a hostname for Ping monitors).

```
void setMachine(
```

```
   String[] names
   String[] values
)
```

setMaxResponseLen(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the maximum amount of data (in bytes) that each of the named monitors should read back from a server (0 = unlimited).

```
void setMaxResponseLen(
   String[] names
   Unsigned Integer[] values
)
```

setNote(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the note for each of the named Monitors.

```
void setNote(
   String[] names
   String[] values
)
```

setPath(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the path that each of the named monitors should use in the test HTTP request.

```
void setPath(
   String[] names
   String[] values
)
```

setProgram(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the name of the program that each named monitor runs.

```
void setProgram(
   String[] names
   String[] values
)
```

setResponseRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the regular expression that each of the named monitors should match against the server response.

```
void setResponseRegex(
   String[] names
   String[] values
)
```

setRtspBodyRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the body regular expression that each of the named monitors' RTSP response must match.

```
void setRtspBodyRegex(
   String[] names
   String[] values
)
```

setRtspPath(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the path that each of the named monitors should use in the test RTSP request.

```
void setRtspPath(
   String[] names
   String[] values
)
```

setRtspStatusRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the status code regular expression that each of the named monitors' RTSP response must match.

```
void setRtspStatusRegex(
   String[] names
   String[] values
)
```

setScope(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the scope of each named monitor.

```
void setScope(
```

```
   String[] names
   Catalog.Monitor.Scope[] values
)
```

setSipBodyRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the body regular expression that that each of the named monitors' SIP response must match.

```
void setSipBodyRegex(
   String[] names
   String[] values
)
```

setSipStatusRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the status code regular expression that that each of the named monitors' SIP response must match.

```
void setSipStatusRegex(
   String[] names
   String[] values
)
```

setSipTransport(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the transport protocol that the monitor will use

```
void setSipTransport(
   String[] names
   Catalog.Monitor.SipTransport[] values
)
```

setStatusRegex(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the status code regular expression that that each of the named monitors' HTTP response must match.

```
void setStatusRegex(
   String[] names
   String[] values
)
```

setTimeout(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the maximum time that an individual instance, of each of the named monitors, is allowed to run for (in seconds).

```
void setTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setType(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the internal monitor type to use for each named monitor.

```
void setType(
   String[] names
   Catalog.Monitor.Type[] values
)
```

setUseSSL(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set whether each of the named monitors can connect using SSL.

```
void setUseSSL(
   String[] names
   Boolean[] values
)
```

setVerbose(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set whether each of the named monitors should emit verbose logging (useful for diagnostics).

```
void setVerbose(
   String[] names
   Boolean[] values
)
```

setWriteString(names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the string that each of the named monitors should write down the TCP connection.

```
void setWriteString(
   String[] names
```

```
   String[] values
)
```

updateProgramArguments(names, argument_names, new_arguments) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Allows arguments for the the specified monitors to be changed. The monitors specified must be of type 'program'.

```
void updateProgramArguments(
   String[] names
   String[][] argument_names
   Catalog.Monitor.Argument[][] new_arguments
)
```

uploadMonitorProgram(name, content) throws InvalidObjectName, DeploymentError

Uploads a monitor program, overwriting the file if it already exists.

```
void uploadMonitorProgram(
   String name
   Binary Data content
)
```

6.6.2 Structures

Catalog.Monitor.Argument

An argument that is added to the command line when the monitor program is run

```
struct Catalog.Monitor.Argument {
   # The name of the argument.
   String name;

   # The value of the argument.
   String value;

   # A description of the argument.
   String description;
}
```

6.6.3 Enumerations

Catalog.Monitor.Scope

```
enum Catalog.Monitor.Scope {
   # Monitor each node separately
```

```
    pernode,

    # Monitor one machine only (pool-wide monitor)
    poolwide
}
```

Catalog.Monitor.SipTransport

```
enum Catalog.Monitor.SipTransport {
    # UDP
    udp,

    # TCP
    tcp
}
```

Catalog.Monitor.Type

```
enum Catalog.Monitor.Type {
    # Ping monitor
    ping,

    # TCP Connect monitor
    connect,

    # HTTP monitor
    http,

    # TCP transaction monitor
    tcp_transaction,

    # External program monitor
    program,

    # SIP monitor
    sip,

    # RTSP monitor
    rtsp
}
```

6.7 Catalog.SSL.Certificates

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/SSL/Certificates/`

The Catalog.SSL.Certificates interface allows management of SSL Certificates which are used for SSL decryption of services. Using this interface, you can create, delete and rename SSL Certificate objects.

6.7.1 Methods

createSelfSignedCertificate(names, details) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput, DeploymentError

Create new self-signed certificates.

```
void createSelfSignedCertificate(
   String[] names
   Catalog.SSL.Certificates.CertificateDetails[] details
)
```

deleteCertificate(names) throws InvalidObjectName, ObjectDoesNotExist

Delete the named certificates.

```
void deleteCertificate(
   String[] names
)
```

getCertificateInfo(names) throws ObjectDoesNotExist

Get the information about the namedcertificates.

```
Certificate[] getCertificateInfo(
   String[] names
)
```

getCertificateNames()

Get the names of the installed certificates.

```
String[] getCertificateNames()
```

getCertificateRequest(names) throws ObjectDoesNotExist

Get Certificate signing requests for the named certificates.

```
String[] getCertificateRequest(
   String[] names
)
```

getRawCertificate(names) throws ObjectDoesNotExist

Get the raw (PEM-encoded) certificates.

```
String[] getRawCertificate(
   String[] names
)
```

importCertificate(names, keys) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput

Create a new certificate, importing the certificate and private key.

```
void importCertificate(
   String[] names
   CertificateFiles[] keys
)
```

renameCertificate(names, new_names) throws InvalidObjectName, ObjectAlreadyExists, ObjectDoesNotExist

Rename the named certificates.

```
void renameCertificate(
   String[] names
   String[] new_names
)
```

setRawCertificate(names, certs) throws ObjectDoesNotExist

Set the (PEM-encoded) certificate. This should be used after getting a Certificate request signed by a certificate authority.

```
void setRawCertificate(
   String[] names
   String[] certs
)
```

6.7.2 Structures

Catalog.SSL.Certificates.CertificateDetails

This structure contains the information used when generating self-signed test certificates.

```
struct Catalog.SSL.Certificates.CertificateDetails {
   # The subject of the new certificate. The common_name of the
   # subject should match the DNS name of the service this
   # certificate is to be used for.
   X509Name subject;

   # The number of days this certificate should be value for
   # (e.g. 365 for 1 years validity)
   Integer valid_days;

   # The size of the generated private key. Use 1024 for normal
   # use, or 2048 for more security
```

```
   Integer key_size;
}
```

Certificate

This structure contains information (such as the subject and issuer) about a certificate.

```
struct Certificate {
   # The version of the X509 Certificate
   Integer version;

   # The serial number of the Certificate
   String serial;

   # The issuer (i.e. who signed it) of the Certificate
   X509Name issuer;

   # The subject (i.e. who it is for) of the Certificate
   X509Name subject;

   # The time the certificate is valid from.
   Time valid_from;

   # The time the certificate is valid to.
   Time valid_to;

   # The modulus of the certificate.
   String modulus;

   # The exponent of the certificate.
   String exponent;

   # Whether the certificate is self-signed (i.e. the issuer is
   # the same as the subject)
   Boolean self_signed;
}
```

CertificateFiles

This structure contains a public certificate and private key. It is used when importing certificates into the traffic manager.

```
struct CertificateFiles {
   # The PEM-encoded public certificate (containing the BEGIN
   # CERTIFICATE and END CERTIFICATE tags)
   String public_cert;

   # The PEM-encoded private key (containing the BEGIN RSA
   # PRIVATE KEY and END RSA PRIVATE KEY tags)
   String private_key;
```

```
}
```

X509Name

This structure contains a representation of an X509 Name object. These are used inside Certificate objects to represent the issuer and subject of the certificate.

```
struct X509Name {
   # The common name (CN). This is usually the name of the site
   # the certificate is issued to (e.g. "secure.example.com")
   String common_name;

   # The two-letter country code.
   String country;

   # The location (town or city).
   String location;

   # The state, this is only needed if the country is 'US'.
   String state;

   # The name of the organisation
   String organisation;

   # The unit inside the organisation
   String unit;

   # An email address. This is usually empty.
   String email;
}
```

6.8 Catalog.SSL.CertificateAuthorities

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/SSL/CertificateAuthorities/`

The Catalog.SSL.CertificateAuthorities interface allows management of SSL Certificate Authorities which are used to authenticate back-end nodes when doing SSL encryption.

6.8.1 Methods

deleteCertificateAuthority(names) throws ObjectDoesNotExist

Delete the named Certificate Authority and associated Revocation list.

```
void deleteCertificateAuthority(
   String[] names
)
```

getCertificateAuthorityInfo(names) throws ObjectDoesNotExist

Get the Certificate Information, and the revoked certificates.

```
Catalog.SSL.CertificateAuthorities.Details[]
getCertificateAuthorityInfo(
   String[] names
)
```

getCertificateAuthorityNames()

Get the names of the configured Certificate Authorities.

```
String[] getCertificateAuthorityNames()
```

getRawCertificate(names) throws ObjectDoesNotExist

Get the raw PEM encoded Certificate for the named Certificate Authorities.

```
String[] getRawCertificate(
   String[] names
)
```

importCRL(crls) throws InvalidInput, ObjectDoesNotExist

Import Certificate Revocation Lists. The associated Certificate Authority certificates should already be imported.

```
void importCRL(
   String[] crls
)
```

importCertificateAuthority(names, certs) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput

Import new Certificate Authorities.

```
void importCertificateAuthority(
   String[] names
   String[] certs
)
```

renameCertificateAuthority(names, new_names) throws InvalidObjectName, ObjectDoesNotExist, ObjectAlreadyExists, InvalidOperation

Rename the named Certificate Authorities.

```
void renameCertificateAuthority(
   String[] names
```

```
    String[] new_names
)
```

6.8.2 Structures

Catalog.SSL.CertificateAuthorities.CRL

This structure contains the information about a Certificate Revocation list.

```
struct Catalog.SSL.CertificateAuthorities.CRL {
    # The time when the CRL was updated
    Time update;

    # The time that the CRL will next be updated.
    Time next_update;

    # The list of revoked certificates
    Catalog.SSL.CertificateAuthorities.RevokedCert[] revoked_certs;
}
```

Catalog.SSL.CertificateAuthorities.Details

This structure contains the information about a Certificate Authority. It contains both the Certificate, and the list of revoked Certificates contained in the associated CRL.

```
struct Catalog.SSL.CertificateAuthorities.Details {
    # The Certificate Authority certificate
    Certificate certificate;

    # If set to 'true' then there is an associated CRL, otherwise
    # the CRL structure contains no useful information
    Boolean have_crl;

    # The associated CRL.
    Catalog.SSL.CertificateAuthorities.CRL crl;
}
```

Catalog.SSL.CertificateAuthorities.RevokedCert

This structure contains the information about a revoked Certificate.

```
struct Catalog.SSL.CertificateAuthorities.RevokedCert {
    # The serial number of the revoked certificate
    String serial;

    # The time that the certificate was revoked
    Time revocation_date;
}
```

Certificate

This structure contains information (such as the subject and issuer) about a certificate.

```
struct Certificate {
   # The version of the X509 Certificate
   Integer version;

   # The serial number of the Certificate
   String serial;

   # The issuer (i.e. who signed it) of the Certificate
   X509Name issuer;

   # The subject (i.e. who it is for) of the Certificate
   X509Name subject;

   # The time the certificate is valid from.
   Time valid_from;

   # The time the certificate is valid to.
   Time valid_to;

   # The modulus of the certificate.
   String modulus;

   # The exponent of the certificate.
   String exponent;

   # Whether the certificate is self-signed (i.e. the issuer is
   # the same as the subject)
   Boolean self_signed;
}
```

CertificateFiles

This structure contains a public certificate and private key. It is used when importing certificates into the traffic manager.

```
struct CertificateFiles {
   # The PEM-encoded public certificate (containing the BEGIN
   # CERTIFICATE and END CERTIFICATE tags)
   String public_cert;

   # The PEM-encoded private key (containing the BEGIN RSA
   # PRIVATE KEY and END RSA PRIVATE KEY tags)
   String private_key;
}
```

X509Name

This structure contains a representation of an X509 Name object. These are used inside Certificate objects to represent the issuer and subject of the certificate.

```
struct X509Name {
   # The common name (CN). This is usually the name of the site
   # the certificate is issued to (e.g. "secure.example.com")
   String common_name;

   # The two-letter country code.
   String country;

   # The location (town or city).
   String location;

   # The state, this is only needed if the country is 'US'.
   String state;

   # The name of the organisation
   String organisation;

   # The unit inside the organisation
   String unit;

   # An email address. This is usually empty.
   String email;
}
```

6.9 Catalog.SSL.ClientCertificates

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/SSL/ClientCertificates/`

The Catalog.SSL.ClientCertificates interface allows management of SSL Client Certificates which are for authentication with back-end nodes when encrypting services. This interfaces allows you to import, retrieve, rename and delete the SSL Client Certificate objects

6.9.1 Methods

deleteClientCertificate(names) throws ObjectDoesNotExist

Delete the named client certificates.

```
void deleteClientCertificate(
   String[] names
)
```

getClientCertificateInfo(names) throws ObjectDoesNotExist

Gets the information about the named client certificates.

```
Certificate[] getClientCertificateInfo(
   String[] names
)
```

getClientCertificateNames()

Get the names of the installed client certificates.

```
String[] getClientCertificateNames()
```

importClientCertificate(names, keys) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput

Import client certificates and associated private keys.

```
void importClientCertificate(
   String[] names
   CertificateFiles[] keys
)
```

renameClientCertificate(names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, DeploymentError

Rename the named client certificates.

```
void renameClientCertificate(
   String[] names
   String[] new_names
)
```

6.9.2 Structures

Certificate

This structure contains information (such as the subject and issuer) about a certificate.

```
struct Certificate {
   # The version of the X509 Certificate
   Integer version;

   # The serial number of the Certificate
   String serial;

   # The issuer (i.e. who signed it) of the Certificate
   X509Name issuer;

   # The subject (i.e. who it is for) of the Certificate
   X509Name subject;
```

```
    # The time the certificate is valid from.
    Time valid_from;

    # The time the certificate is valid to.
    Time valid_to;

    # The modulus of the certificate.
    String modulus;

    # The exponent of the certificate.
    String exponent;

    # Whether the certificate is self-signed (i.e. the issuer is
    # the same as the subject)
    Boolean self_signed;
}
```

CertificateFiles

This structure contains a public certificate and private key. It is used when importing certificates into the traffic manager.

```
struct CertificateFiles {
    # The PEM-encoded public certificate (containing the BEGIN
    # CERTIFICATE and END CERTIFICATE tags)
    String public_cert;

    # The PEM-encoded private key (containing the BEGIN RSA
    # PRIVATE KEY and END RSA PRIVATE KEY tags)
    String private_key;
}
```

X509Name

This structure contains a representation of an X509 Name object. These are used inside Certificate objects to represent the issuer and subject of the certificate.

```
struct X509Name {
    # The common name (CN). This is usually the name of the site
    # the certificate is issued to (e.g. "secure.example.com")
    String common_name;

    # The two-letter country code.
    String country;

    # The location (town or city).
    String location;

    # The state, this is only needed if the country is 'US'.
    String state;
```

```
   # The name of the organisation
   String organisation;

   # The unit inside the organisation
   String unit;

   # An email address. This is usually empty.
   String email;
}
```

6.10 Catalog.Protection

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/Protection/`

The Catalog.Protection interface allows management of Service Protection classes. Using this interface, you can create, delete and rename Protection classes, and manage their configuration.

6.10.1 Methods

addAllowedAddresses(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add new IP addresses and CIDR IP subnets to the list of machines that are always allowed access.

```
void addAllowedAddresses(
   String[] class_names
   String[][] values
)
```

addBannedAddresses(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add new IP addresses and CIDR IP subnets to the list of machines that aren't allowed access.

```
void addBannedAddresses(
   String[] class_names
   String[][] values
)
```

addProtection(class_names) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add new Protection classes.

```
void addProtection(
   String[] class_names
)
```

copyProtection(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Copy the named Protection classes.

```
void copyProtection(
   String[] class_names
   String[] new_names
)
```

deleteProtection(class_names) throws ObjectInUse, ObjectDoesNotExist, DeploymentError

Delete the named Protection classes.

```
void deleteProtection(
   String[] class_names
)
```

getAllowedAddresses(class_names) throws ObjectDoesNotExist

Get the list of IP addresses and CIDR IP subnets that are always allowed access.

```
String[][] getAllowedAddresses(
   String[] class_names
)
```

getBannedAddresses(class_names) throws ObjectDoesNotExist

Get the list of IP addresses and CIDR IP subnets that aren't allowed access.

```
String[][] getBannedAddresses(
   String[] class_names
)
```

getDebug(class_names) throws ObjectDoesNotExist

Get whether the service protection classes are in debug mode. When in debug mode, verbose log messages are written.

```
Boolean[] getDebug(
   String[] class_names
)
```

getEnabled(class_names) throws ObjectDoesNotExist

Get whether the service protection classes are enabled.

```
Boolean[] getEnabled(
   String[] class_names
)
```

getHTTPCheckRfc2396(class_names) throws ObjectDoesNotExist

Get whether requests with poorly-formed URLs (as specified in RFC 2396) should be rejected.

```
Boolean[] getHTTPCheckRfc2396(
   String[] class_names
)
```

getHTTPRejectBinary(class_names) throws ObjectDoesNotExist

Get whether requests containing binary data (after decoding) should be rejected.

```
Boolean[] getHTTPRejectBinary(
   String[] class_names
)
```

getHTTPSendErrorPage(class_names) throws ObjectDoesNotExist

Get whether an HTTP error message should be sent when a connection is dropped, rather than just dropping the connection.

```
Boolean[] getHTTPSendErrorPage(
   String[] class_names
)
```

getLogInterval(class_names) throws ObjectDoesNotExist

Get the interval between logging service protection messages (in seconds).

```
Unsigned Integer[] getLogInterval(
   String[] class_names
)
```

getMax10Connections(class_names) throws ObjectDoesNotExist

Get the maximum number of simultaneous connections allowed from the 10 busiest IP addresses.

```
Unsigned Integer[] getMax10Connections(
   String[] class_names
```

```
)
```

getMax1Connections(class_names) throws ObjectDoesNotExist

Get the maximum number of simultaneous connections allowed from an individual IP address.

```
Unsigned Integer[] getMax1Connections(
   String[] class_names
)
```

getMaxConnectionRate(class_names) throws ObjectDoesNotExist

Get the maximum number of connections and HTTP keepalive requests allowed from 1 IP address in the 'rate_timer' interval (0 means unlimited).

```
Unsigned Integer[] getMaxConnectionRate(
   String[] class_names
)
```

getMaxHTTPBodyLength(class_names) throws ObjectDoesNotExist

Get the maximum size of the HTTP body data (in bytes, 0 means no limit).

```
Unsigned Integer[] getMaxHTTPBodyLength(
   String[] class_names
)
```

getMaxHTTPHeaderLength(class_names) throws ObjectDoesNotExist

Get the maximum size of a single HTTP header (in bytes, 0 means no limit).

```
Unsigned Integer[] getMaxHTTPHeaderLength(
   String[] class_names
)
```

getMaxHTTPRequestLength(class_names) throws ObjectDoesNotExist

Get the maximum size of all the HTTP request headers (in bytes, 0 means no limit).

```
Unsigned Integer[] getMaxHTTPRequestLength(
   String[] class_names
)
```

getMaxHTTPURLLength(class_names) throws ObjectDoesNotExist

Get the maximum size of the request URL (in bytes, 0 means no limit).

```
Unsigned Integer[] getMaxHTTPURLLength(
   String[] class_names
)
```

getMinConnections(class_names) throws ObjectDoesNotExist

Get the number of simultaneous connections that are always allowed from each IP address (0 means unlimited).

```
Unsigned Integer[] getMinConnections(
   String[] class_names
)
```

getNote(class_names) throws ObjectDoesNotExist

Get the note for each of the named Protection classes

```
String[] getNote(
   String[] class_names
)
```

getProtectionNames()

Get the names of all the configured Protection classes.

```
String[] getProtectionNames()
```

getRateTimer(class_names) throws ObjectDoesNotExist

Get how frequently the max_connection_rate is assessed. For example, a value of 1 second will impose a limit of max connections/second; a value of 60 will impose a limit of max connections/minute controlling how our connection rates are calculated.

```
Unsigned Integer[] getRateTimer(
   String[] class_names
)
```

getRule(class_names) throws ObjectDoesNotExist

Get the TrafficScript rule to be applied to all connections.

```
String[] getRule(
   String[] class_names
)
```

getTesting(class_names) throws ObjectDoesNotExist

Get whether the service protection classes are in testing mode. When in testing mode the class logs when a connection would be dropped, but it allows all connections through.

```
Boolean[] getTesting(
   String[] class_names
)
```

removeAllowedAddresses(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Remove IP addresses and CIDR IP subnets from the list of machines that are always allowed access.

```
void removeAllowedAddresses(
   String[] class_names
   String[][] values
)
```

removeBannedAddresses(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Remove IP addresses and CIDR IP subnets from the list of machines that aren't allowed access.

```
void removeBannedAddresses(
   String[] class_names
   String[][] values
)
```

renameProtection(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename the named Protection classes.

```
void renameProtection(
   String[] class_names
   String[] new_names
)
```

setAllowedAddresses(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the list of IP addresses and CIDR IP subnets that are always allowed access.

```
void setAllowedAddresses(
   String[] class_names
   String[][] values
```

```
)
```

setBannedAddresses(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the list of IP addresses and CIDR IP subnets that aren't allowed access.

```
void setBannedAddresses(
   String[] class_names
   String[][] values
)
```

setDebug(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether the service protection classes are in debug mode. When in debug mode, verbose log messages are written.

```
void setDebug(
   String[] class_names
   Boolean[] values
)
```

setEnabled(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether the service protection classes are enabled.

```
void setEnabled(
   String[] class_names
   Boolean[] values
)
```

setHTTPCheckRfc2396(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether requests with poorly-formed URLs (as specified in RFC 2396) should be rejected.

```
void setHTTPCheckRfc2396(
   String[] class_names
   Boolean[] values
)
```

setHTTPRejectBinary(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether requests containing binary data (after decoding) should be rejected.

```
void setHTTPRejectBinary(
   String[] class_names
   Boolean[] values
)
```

setHTTPSendErrorPage(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether an HTTP error message should be sent when a connection is dropped, rather than just dropping the connection.

```
void setHTTPSendErrorPage(
   String[] class_names
   Boolean[] values
)
```

setLogInterval(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the interval between logging service protection messages (in seconds).

```
void setLogInterval(
   String[] class_names
   Unsigned Integer[] values
)
```

setMax10Connections(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum number of simultaneous connections allowed from the 10 busiest IP addresses.

```
void setMax10Connections(
   String[] class_names
   Unsigned Integer[] values
)
```

setMax1Connections(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum number of simultaneous connections allowed from an individual IP address.

```
void setMax1Connections(
```

```
   String[] class_names
   Unsigned Integer[] values
)
```

setMaxConnectionRate(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum number of connections and HTTP keepalive requests allowed from 1 IP address in the 'rate_timer' interval (0 means unlimited).

```
void setMaxConnectionRate(
   String[] class_names
   Unsigned Integer[] values
)
```

setMaxHTTPBodyLength(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of the HTTP body data (in bytes, 0 means no limit).

```
void setMaxHTTPBodyLength(
   String[] class_names
   Unsigned Integer[] values
)
```

setMaxHTTPHeaderLength(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of a single HTTP header (in bytes, 0 means no limit).

```
void setMaxHTTPHeaderLength(
   String[] class_names
   Unsigned Integer[] values
)
```

setMaxHTTPRequestLength(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of all the HTTP request headers (in bytes, 0 means no limit).

```
void setMaxHTTPRequestLength(
   String[] class_names
   Unsigned Integer[] values
)
```

setMaxHTTPURLLength(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of the request URL (in bytes, 0 means no limit).

```
void setMaxHTTPURLLength(
   String[] class_names
   Unsigned Integer[] values
)
```

setMinConnections(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the number of simultaneous connections that are always allowed from each IP address (0 means unlimited).

```
void setMinConnections(
   String[] class_names
   Unsigned Integer[] values
)
```

setNote(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the note for each of the named Protection classes

```
void setNote(
   String[] class_names
   String[] values
)
```

setRateTimer(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set how frequently the max_connection_rate is assessed. For example, a value of 1 second will impose a limit of max connections/second; a value of 60 will impose a limit of max connections/minute controlling how our connection rates are calculated.

```
void setRateTimer(
   String[] class_names
   Unsigned Integer[] values
)
```

setRule(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the TrafficScript rule to be applied to all connections.

```
void setRule(
```

```
   String[] class_names
   String[] values
)
```

setTesting(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether the service protection classes are in testing mode. When in testing mode the class logs when a connection would be dropped, but it allows all connections through.

```
void setTesting(
   String[] class_names
   Boolean[] values
)
```

6.11 Catalog.Persistence

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/Persistence/`

The Catalog.Persistence interface allows management of Persistence classes. Using this interface, you can create, delete and rename persistence classes, and manage their configuration.

6.11.1 Methods

addPersistence(class_names) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add new persistence classes.

```
void addPersistence(
   String[] class_names
)
```

copyPersistence(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Copy the named persistence classes.

```
void copyPersistence(
   String[] class_names
   String[] new_names
)
```

deletePersistence(class_names) throws ObjectDoesNotExist, ObjectInUse, DeploymentError

Delete the named persistence classes.

```
void deletePersistence(
   String[] class_names
)
```

getCookie(class_names) throws ObjectDoesNotExist

Get the name of the cookie used to track session persistence.

```
String[] getCookie(
   String[] class_names
)
```

getDelete(class_names) throws ObjectDoesNotExist

Get whether the session should be deleted if a failure occurs.

```
Boolean[] getDelete(
   String[] class_names
)
```

getFailureMode(class_names) throws ObjectDoesNotExist

Get the action that should be taken if the session data is invalid or the node specified cannot be contacted.

```
Catalog.Persistence.FailureMode[] getFailureMode(
   String[] class_names
)
```

getNote(class_names) throws ObjectDoesNotExist

Get the note for each of the named Session Persistence classes.

```
String[] getNote(
   String[] class_names
)
```

getPersistenceNames()

Get the names of all the configured persistence classes.

```
String[] getPersistenceNames()
```

getType(class_names) throws ObjectDoesNotExist

Gets the session method type.

```
Catalog.Persistence.Type[] getType(
   String[] class_names
)
```

getUrl(class_names) throws ObjectDoesNotExist

Get the URL to send to clients if the session persistence is configured to redirect users when a node dies.

```
String[] getUrl(
   String[] class_names
)
```

renamePersistence(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename the named persistence classes.

```
void renamePersistence(
   String[] class_names
   String[] new_names
)
```

setCookie(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the name of the cookie used to track session persistence.

```
void setCookie(
   String[] class_names
   String[] values
)
```

setDelete(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether the session should be deleted if a failure occurs.

```
void setDelete(
   String[] class_names
   Boolean[] values
)
```

setFailureMode(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the action that should be taken if the session data is invalid or the node specified cannot be contacted.

```
void setFailureMode(
   String[] class_names
   Catalog.Persistence.FailureMode[] values
)
```

setNote(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the note for each of the named Session Persistence classes.

```
void setNote(
   String[] class_names
   String[] values
)
```

setType(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Sets the session method type.

```
void setType(
   String[] class_names
   Catalog.Persistence.Type[] values
)
```

setUrl(class_names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the URL to send to clients if the session persistence is configured to redirect users when a node dies.

```
void setUrl(
   String[] class_names
   String[] values
)
```

6.11.2 Enumerations

Catalog.Persistence.FailureMode

```
enum Catalog.Persistence.FailureMode {
   # Choose a new node to use
```

```
    newnode,

    # Redirect the user to a given URL
    url,

    # Close the connection (using error_file on Pools > Edit >
    # Connection Management)
    close
}
```

Catalog.Persistence.Type

```
enum Catalog.Persistence.Type {
    # IP-based persistence
    ip,

    # Universal session persistence
    universal,

    # Named Node session persistence
    named,

    # Transparent session affinity
    transparent,

    # Monitor application cookies
    monitor-cookies,

    # J2EE session persistence
    j2ee,

    # ASP and ASP.NET session persistence
    asp,

    # X-Zeus-Backend cookies
    x-zeus,

    # SSL Session ID persistence
    ssl,

    # Deprecated. Use 'monitor-cookies' instead.
    kipper,

    # Deprecated. Use 'transparent' instead.
    sardine
}
```

6.12 Catalog.Bandwidth

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/Bandwidth/`

The Catalog.Bandwidth interface allows management of Bandwidth classes. Using this interface, you can create, delete and rename bandwidth classes, and manage their configuration.

6.12.1 Methods

addBandwidth(class_names) throws InvalidOperation, ObjectAlreadyExists, InvalidObjectName, LicenseError, DeploymentError

Add new bandwidth classes.

```
void addBandwidth(
   String[] class_names
)
```

copyBandwidth(class_names, new_names) throws InvalidOperation, ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, LicenseError

Copy the named bandwidth classes.

```
void copyBandwidth(
   String[] class_names
   String[] new_names
)
```

deleteBandwidth(class_names) throws ObjectDoesNotExist, ObjectInUse, LicenseError, DeploymentError

Delete the named bandwidth classes.

```
void deleteBandwidth(
   String[] class_names
)
```

getBandwidthNames() throws LicenseError

Get the names of all the configured bandwidth classes.

```
String[] getBandwidthNames()
```

getMaximum(class_names) throws ObjectDoesNotExist, LicenseError

Get the maximum bandwidth, in kbits/second.

```
Unsigned Integer[] getMaximum(
   String[] class_names
)
```

getNote(class_names) throws ObjectDoesNotExist, LicenseError

Get the note for each of the named Bandwidth classes.

```
String[] getNote(
   String[] class_names
)
```

getSharing(class_names) throws ObjectDoesNotExist, LicenseError

Get the bandwidth sharing mode

```
Catalog.Bandwidth.Sharing[] getSharing(
   String[] class_names
)
```

renameBandwidth(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation, LicenseError

Rename the named bandwidth classes.

```
void renameBandwidth(
   String[] class_names
   String[] new_names
)
```

setMaximum(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the maximum bandwidth, in kbits/second.

```
void setMaximum(
   String[] class_names
   Unsigned Integer[] values
)
```

setNote(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the note for each of the named Bandwidth classes.

```
void setNote(
   String[] class_names
```

```
   String[] values
)
```

setSharing(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the bandwidth sharing mode

```
void setSharing(
   String[] class_names
   Catalog.Bandwidth.Sharing[] values
)
```

6.12.2 Enumerations

Catalog.Bandwidth.Sharing

```
enum Catalog.Bandwidth.Sharing {
   # Each connection can use the maximum rate
   connection,

   # Bandwidth is shared per traffic manager
   machine,

   # Bandwidth is shared across all traffic managers
   cluster
}
```

6.13 Catalog.SLM

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/SLM/`

The Catalog.SLM interface allows management of Service Level Monitoring classes. Using this interface, you can create, delete and rename SLM classes, and manage their configuration.

6.13.1 Methods

addSLM(class_names) throws InvalidObjectName, ObjectAlreadyExists, DeploymentError, LicenseError

Add new SLM classes.

```
void addSLM(
   String[] class_names
)
```

copySLM(class_names, new_names) throws ObjectAlreadyExists, InvalidObjectName, ObjectDoesNotExist, DeploymentError, LicenseError

Copy the named SLM classes.

```
void copySLM(
   String[] class_names
   String[] new_names
)
```

deleteSLM(class_names) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, LicenseError

Delete the named SLM classes.

```
void deleteSLM(
   String[] class_names
)
```

getNote(class_names) throws ObjectDoesNotExist, LicenseError

Get the note for each of the named SLM classes.

```
String[] getNote(
   String[] class_names
)
```

getResponseTime(class_names) throws ObjectDoesNotExist, LicenseError

Get the time limit for a response to conform (in milliseconds).

```
Unsigned Integer[] getResponseTime(
   String[] class_names
)
```

getSLMNames() throws LicenseError

Get the names of all the configured SLM classes.

```
String[] getSLMNames()
```

getSeriousThreshold(class_names) throws ObjectDoesNotExist, LicenseError

Get the percentage of conforming responses below which a serious error will be emitted.

```
Unsigned Integer[] getSeriousThreshold(
   String[] class_names
)
```

getWarningThreshold(class_names) throws ObjectDoesNotExist, LicenseError

Get the percentage of conforming responses below which a warning message will be triggered.

```
Unsigned Integer[] getWarningThreshold(
   String[] class_names
)
```

renameSLM(class_names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, InvalidOperation, DeploymentError, LicenseError

Rename the named SLM classes.

```
void renameSLM(
   String[] class_names
   String[] new_names
)
```

setNote(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the note for each of the named SLM classes.

```
void setNote(
   String[] class_names
   String[] values
)
```

setResponseTime(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the time limit for a response to conform (in milliseconds).

```
void setResponseTime(
   String[] class_names
   Unsigned Integer[] values
)
```

setSeriousThreshold(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the percentage of conforming responses below which a serious error will be emitted.

```
void setSeriousThreshold(
   String[] class_names
   Unsigned Integer[] values
)
```

setWarningThreshold(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the percentage of conforming responses below which a warning message will be triggered.

```
void setWarningThreshold(
   String[] class_names
   Unsigned Integer[] values
)
```

6.14 Catalog.Rate

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/Rate/`

The Catalog.Rate interface allows management of Rate classes. Using this interface, you can create, delete and rename rate classes, and manage their configuration.

6.14.1 Methods

addRate(class_names) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add new rate classes.

```
void addRate(
   String[] class_names
)
```

copyRate(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Copy the named rate classes.

```
void copyRate(
   String[] class_names
   String[] new_names
)
```

deleteRate(class_names) throws ObjectInUse, ObjectDoesNotExist, DeploymentError

Delete the named rate classes.

```
void deleteRate(
   String[] class_names
)
```

getMaxRatePerMinute(class_names) throws ObjectDoesNotExist

Get the maximum rate at which requests are allowed to be processed, in requests per minute.

```
Unsigned Integer[] getMaxRatePerMinute(
   String[] class_names
)
```

getMaxRatePerSecond(class_names) throws ObjectDoesNotExist

Get the maximum rate at which requests are allowed to be processed, in requests per second.

```
Unsigned Integer[] getMaxRatePerSecond(
   String[] class_names
)
```

getNote(class_names) throws ObjectDoesNotExist

Get the note for each of the named Rate classes.

```
String[] getNote(
   String[] class_names
)
```

getRateNames()

Get the names of all the configured rate classes.

```
String[] getRateNames()
```

renameRate(class_names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename the named rate classes.

```
void renameRate(
   String[] class_names
   String[] new_names
)
```

setMaxRatePerMinute(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum rate at which requests are allowed to be processed, in requests per minute.

```
void setMaxRatePerMinute(
   String[] class_names
   Unsigned Integer[] values
)
```

setMaxRatePerSecond(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum rate at which requests are allowed to be processed, in requests per second.

```
void setMaxRatePerSecond(
   String[] class_names
   Unsigned Integer[] values
)
```

setNote(class_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the note for each of the named Rate classes.

```
void setNote(
   String[] class_names
   String[] values
)
```

6.15 Catalog.JavaExtension

URI: `http://soap.zeus.com/zxtm/1.0/Catalog/JavaExtension/`

The Catalog.JavaExtension interface allows management of Java Extensions. Using this interface you can retrieve information on each extension in the system, and set the initialisation properties to alter their behaviour.

6.15.1 Methods

addProperties(class_names, properties) throws LicenseError, ObjectDoesNotExist, InvalidInput

Adds initialisation properties for each of the specified extensions.

```
void addProperties(
   String[] class_names
   Catalog.JavaExtension.Property[][] properties
)
```

deleteJavaExtensionFile(names) throws LicenseError, ObjectDoesNotExist, ObjectInUse

Delete the named Java Extension files.

```
void deleteJavaExtensionFile(
   String[] names
)
```

downloadJavaExtensionFile(name) throws LicenseError, ObjectDoesNotExist

Download the named Java Extension File.

```
Binary Data downloadJavaExtensionFile(
   String name
)
```

editProperties(class_names, properties_being_edited, properties) throws LicenseError, ObjectDoesNotExist, InvalidInput

Edits the initialisation properties for each of the specified extensions.

```
void editProperties(
   String[] class_names
   String[][] properties_being_edited
   Catalog.JavaExtension.Property[][] properties
)
```

getExtensionClassNames() throws LicenseError

Gets the class names of all the extensions currently in the system.

```
String[] getExtensionClassNames()
```

getExtensionErrors(class_names) throws LicenseError, ObjectDoesNotExist

Gets the errors for each of the specified extensions.

```
String[][] getExtensionErrors(
   String[] class_names
)
```

getExtensionInfo(class_names) throws LicenseError, ObjectDoesNotExist

Gets information on each of the specified extensions.

```
Catalog.JavaExtension.Info[] getExtensionInfo(
   String[] class_names
)
```

getJavaExtensionFileNames() throws LicenseError

Get the names of Java Extension files on the traffic manager. This list includes files that contain Java Extension and non-Java Extension files, such as other .jar files.

```
String[] getJavaExtensionFileNames()
```

getProperties(class_names) throws LicenseError, ObjectDoesNotExist

Gets the initialisation properties for each of the specified extensions.

```
Catalog.JavaExtension.Property[][] getProperties(
   String[] class_names
)
```

removeProperties(class_names, prop_names) throws LicenseError, ObjectDoesNotExist, InvalidInput

Removes initialisation properties for each of the specified extensions.

```
void removeProperties(
   String[] class_names
   String[][] prop_names
)
```

uploadJavaExtensionFile(name, content) throws InvalidObjectName, LicenseError, InvalidInput

Uploads a new file that may contain a Java Extension. This will overwrite the file if it already exists.

```
void uploadJavaExtensionFile(
   String name
   Binary Data content
)
```

6.15.2 Structures

Catalog.JavaExtension.Info

This structure contains basic information about a Java Extension in the catalog.

```
struct Catalog.JavaExtension.Info {
   # The Java class name of the extension.
   String class_name;

   # The location of the Java extension class.
```

```
   String path;

   # The virtual servers that use this extension.
   String[] virtual_servers;

   # The rules that use this extension.
   String[] rules;
}
```

Catalog.JavaExtension.Property

Represents an initialisation property for an extension.

```
struct Catalog.JavaExtension.Property {
   # The name of this property
   String name;

   # The value of this property
   String value;
}
```

6.16 GlobalSettings

URI: `http://soap.zeus.com/zxtm/1.0/GlobalSettings/`

The Global Settings interface allows management of the traffic manager settings.

6.16.1 Methods

addFlipperFrontendCheckAddresses(values) throws InvalidInput, DeploymentError

Add new IP addresses to the list that should be used to check front-end connectivity

```
void addFlipperFrontendCheckAddresses(
   String[] values
)
```

getASPSessionCacheSize()

Get the maximum number of entries in the ASP session cache.

```
Unsigned Integer getASPSessionCacheSize()
```

getAcceptingDelay()

Get how often each traffic manager child process checks whether it should be accepting new connections.

```
Unsigned Integer getAcceptingDelay()
```

getAlertEmailInterval()

Get the length of time between alert emails, in seconds. Several alert messages will be stored up and sent in one email.

```
Unsigned Integer getAlertEmailInterval()
```

getAlertEmailMaxAttempts()

Get the number of times to attempt sending an email before giving up.

```
Unsigned Integer getAlertEmailMaxAttempts()
```

getBackendKeepaliveTimeout()

getBackendKeepaliveTimeout is deprecated, please use getIdleConnectionTimeout instead.

```
Unsigned Integer getBackendKeepaliveTimeout()
```

getBandwidthSharing()

This method is now obsolete and is replaced by Catalog.Bandwidth.getSharing.

```
Boolean getBandwidthSharing()
```

getChunkSize()

Get the default chunk size for reading and writing data, in bytes.

```
Unsigned Integer getChunkSize()
```

getClientFirstOpt()

Get whether client-first network socket optimisations should be used.

```
Boolean getClientFirstOpt()
```

getControlAllowHosts()

Get the hosts that are allowed to contact the internal administration port on each traffic manager.

```
String getControlAllowHosts()
```

getDNSCacheExpiryTime()

Get the time entries are stored in the DNS cache for, in seconds.

```
Unsigned Integer getDNSCacheExpiryTime()
```

getDNSCacheNegativeExpiryTime()

Get the time failed lookups are stored in the DNS cache for, in seconds.

```
Unsigned Integer getDNSCacheNegativeExpiryTime()
```

getDNSCacheSize()

Get the maximum number of entries in the DNS cache.

```
Unsigned Integer getDNSCacheSize()
```

getDNSTimeout()

Get the timeout for receiving a response from a DNS Server, in seconds.

```
Unsigned Integer getDNSTimeout()
```

getDeadTime()

This method is now obsolete and is replaced by Pool.getNodeFailTime.

```
Unsigned Integer getDeadTime()
```

getEC2AccessKeyID()

Get the Access Key ID used for interacting with the EC2 API.

```
String getEC2AccessKeyID()
```

getErrorLevel()

Get the minimum severity of events that should be logged to disk.

```
GlobalSettings.ErrorLevel getErrorLevel()
```

getErrorLogFile()

Get the filename that errors are logged to.

```
String getErrorLogFile()
```

getFTPDataBindLow()

Get whether your traffic manager should permit use of FTP data connection source ports lower than 1024. If 'No' your traffic manager can completely drop root privileges, if 'Yes' some or all privileges may be retained in order to bind to low ports.

```
Boolean getFTPDataBindLow()
```

getFlipperArpCount()

Get the number of ARP packets each traffic manager sends when an interface is raised.

```
Unsigned Integer getFlipperArpCount()
```

getFlipperAutofailback()

Get whether Traffic IPs should automatically failback to recovered machines.

```
Boolean getFlipperAutofailback()
```

getFlipperFrontendCheckAddresses()

Get the IP addresses that should be used to check front-end connectivity.

```
String[] getFlipperFrontendCheckAddresses()
```

getFlipperHeartbeatMethod()

Get the method used to exchange cluster heartbeat messages.

```
GlobalSettings.FlipperHeartbeatMethod getFlipperHeartbeatMethod()
```

getFlipperMonitorInterval()

Get how frequently (in milliseconds) each traffic manager checks and announces its connectivity.

```
Unsigned Integer getFlipperMonitorInterval()
```

getFlipperMonitorTimeout()

Get how long (in seconds) each traffic manager waits for a response from its connectivity tests or from other traffic managers before registering a failure.

```
Unsigned Integer getFlipperMonitorTimeout()
```

getFlipperMulticastAddress()

Get the multicast address and port used to announce connectivity (e.g. 239.100.1.1:9090).

```
String getFlipperMulticastAddress()
```

getFlipperUnicastPort()

Get the unicast UDP port used to announce connectivity (e.g. 9090)

```
Unsigned Integer getFlipperUnicastPort()
```

getFlipperUseBindip()

Get whether the heartbeat messages used for fault tolerance are only sent over the management network.

```
Boolean getFlipperUseBindip()
```

getFlipperVerbose()

Get whether the traffic manager should logs all the connectivity tests.

```
Boolean getFlipperVerbose()
```

getHistoricalTrafficDays()

Get the length of time historical traffic information is kept for, in days (0=keep indefinitely).

```
Unsigned Integer getHistoricalTrafficDays()
```

getIPSessionCacheSize()

Get the maximum number of entries in the IP session cache.

```
Unsigned Integer getIPSessionCacheSize()
```

getIdleConnectionTimeout()

Get how long unused HTTP keepalive connections should be kept before being discarded, in seconds.

```
Unsigned Integer getIdleConnectionTimeout()
```

getJ2EESessionCacheSize()

Get the maximum number of entries in the J2EE session cache.

```
Unsigned Integer getJ2EESessionCacheSize()
```

getJavaClasspath()

Get extra Java CLASSPATH settings required for servlets.

```
String getJavaClasspath()
```

getJavaCommand()

Get the command (and arguments) used to start Java.

```
String getJavaCommand()
```

getJavaEnabled()

Get whether to enable Java support.

```
Boolean getJavaEnabled()
```

getJavaLib()

Get the location of the java library directory

```
String getJavaLib()
```

getJavaMaxConns()

Get the maximum number of Java threads

```
Unsigned Integer getJavaMaxConns()
```

getJavaSessionAge()

Get the default maximum age of Java session persistence

```
Unsigned Integer getJavaSessionAge()
```

getListenQueueSize()

Get the size of the listen queue for managing incoming connections.

```
Unsigned Integer getListenQueueSize()
```

getLogFlushFlushTime()

Get the length of time to wait before flushing the request log files for each virtual server, in seconds.

```
Unsigned Integer getLogFlushFlushTime()
```

getLogInterval()

Get the length of time between log messages for log intensive features e.g. SLM, in seconds.

```
Unsigned Integer getLogInterval()
```

getLogRate()

Get is the maximum number of connection errors logged per second.

```
Unsigned Integer getLogRate()
```

getLogReopenTime()

Get the length of time to wait before re-opening request log files, to handle log file rotation, in seconds.

```
Unsigned Integer getLogReopenTime()
```

getMaxAccepting()

Get how many traffic manager child processes accept new connections.

```
Unsigned Integer getMaxAccepting()
```

getMaxIdleConnections()

Get the maximum number of unused HTTP keepalive connections to all nodes that should maintained for re-use.

```
Unsigned Integer getMaxIdleConnections()
```

getMaxKeepalives()

getMaxKeepalives is deprecated, please use getMaxIdleConnections instead.

```
Unsigned Integer getMaxKeepalives()
```

getMaxRetries()

This method is now obsolete and is replaced by Pool.getNodeConnectionAttempts.

```
Unsigned Integer getMaxRetries()
```

getMaximumFDCount()

Get the maximum number of file descriptors that your traffic manager will allocate

```
Unsigned Integer getMaximumFDCount()
```

getMonitorNumNodes()

Get the maximum number of nodes that can be monitored.

```
Unsigned Integer getMonitorNumNodes()
```

getMultipleAccept()

Get whether your traffic manager should try and read multiple new connections each time a new client connects.

```
Boolean getMultipleAccept()
```

getNodeConnectionAttempts()

This method is now obsolete and is replaced by Pool.getNodeConnectionAttempts.

```
Unsigned Integer getNodeConnectionAttempts()
```

getNodeFailTime()

This method is now obsolete and is replaced by Pool.getNodeFailTime.

```
Unsigned Integer getNodeFailTime()
```

getRateClassLimit()

Get the maximum number of Rate classes allowed.

```
Unsigned Integer getRateClassLimit()
```

getRecentConns()

Get how many recent connections each traffic manager process should record for the 'active connections' list.

```
Unsigned Integer getRecentConns()
```

getSLMClassLimit()

Get the maximum number of SLM classes allowed.

```
Unsigned Integer getSLMClassLimit()
```

getSNMPUserCounters()

Get the number of user defined SNMP counters.

```
Unsigned Integer getSNMPUserCounters()
```

getSSL3Ciphers()

Get the list of configured SSL ciphers (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s).

```
String getSSL3Ciphers()
```

getSSLDFailureCount()

getSSLDFailureCount is deprecated, please use getSSLHardwareFailureCount instead.

```
Unsigned Integer getSSLDFailureCount()
```

getSSLDPKCS11Lib()

getSSLDPKCS11Lib is deprecated, please use getSSLHardwarePKCS11Lib instead.

```
String getSSLDPKCS11Lib()
```

getSSLHardwareAccelerator()

Get whether your traffic manager should always attempt to use SSL hardware.

```
Boolean getSSLHardwareAccelerator()
```

getSSLHardwareFailureCount()

Get the number of consecutive failures from the SSL hardware that will be tolerated before your traffic manager tries to log in again.

```
Unsigned Integer getSSLHardwareFailureCount()
```

getSSLHardwarePKCS11Lib()

Get the location of the PKCS#11 library supplied by your hardware vendor.

```
String getSSLHardwarePKCS11Lib()
```

getSSLHardwareType()

Get the device driver library name.

```
GlobalSettings.SSLHardwareType getSSLHardwareType()
```

getSSLSessionCacheSize()

Get the maximum number of entries in the SSL session cache. This is used to provide persistence based on SSL session IDs.

```
Unsigned Integer getSSLSessionCacheSize()
```

getSSLSessionIDCacheExpiryTime()

Get the length of time that SSL session IDs are stored, in seconds.

```
Unsigned Integer getSSLSessionIDCacheExpiryTime()
```

getSSLSessionIDCacheSize()

Get the number of entries in the SSL session ID cache.

```
Unsigned Integer getSSLSessionIDCacheSize()
```

getSSLSupportSSL2()

Get whether SSLv2 support is enabled.

```
Boolean getSSLSupportSSL2()
```

getSSLSupportSSL3()

Get whether SSLv3 support is enabled.

```
Boolean getSSLSupportSSL3()
```

getSSLSupportTLS1()

Get whether TLSv1 support is enabled.

```
Boolean getSSLSupportTLS1()
```

getSSLSupportTLS11()

Get whether TLSv1.1 support is enabled.

```
Boolean getSSLSupportTLS11()
```

getSoapIdleMinutes()

Get the number of minutes the SOAP server remain idle before exiting

```
Unsigned Integer getSoapIdleMinutes()
```

getSocketOptimizations()

Get whether potential network socket optimisations should be used.

```
GlobalSettings.SocketOptimizations getSocketOptimizations()
```

getSsldAccel()

getSsldAccel is deprecated, please use getSSLHardwareAccelerator instead.

```
Boolean getSsldAccel()
```

getSsldLibrary()

getSsldLibrary is deprecated, please use getSSLHardwareType instead.

```
GlobalSettings.SsldLibrary getSsldLibrary()
```

getStateSyncTime()

Get how often the cache state is propagated to other traffic managers in the cluster, in seconds.

```
Unsigned Integer getStateSyncTime()
```

getStateSyncTimeout()

Get the timeout for state propagation between cluster members, in seconds

```
Unsigned Integer getStateSyncTimeout()
```

getSystemReadBufferSize()

Get the size of the operating system's read buffer, in bytes (0 means use the system default).

```
Unsigned Integer getSystemReadBufferSize()
```

getSystemWriteBufferSize()

Get the size of the operating system's write buffer, in bytes (0 means use the system default).

```
Unsigned Integer getSystemWriteBufferSize()
```

getTrafficScriptMemoryWarning()

Get the amount of buffered network data a TrafficScript rule can buffer before a warning is logged, in bytes.

```
Unsigned Integer getTrafficScriptMemoryWarning()
```

getTrafficscriptDataSize()

Get the maximum size of the TrafficScript shared data pool (specified as a percentage of system RAM, e.g. '5%', or an absolute size, e.g. 200MB)

```
String getTrafficscriptDataSize()
```

getTrafficscriptMaxInstr()

Get the maximum number of instructions a TrafficScript rule will run before being aborted.

```
Unsigned Integer getTrafficscriptMaxInstr()
```

getTrafficscriptRegexCacheSize()

Get the number of regular expressions to cache

```
Unsigned Integer getTrafficscriptRegexCacheSize()
```

getTrafficscriptVariablePoolUse()

Get whether the 'pool.use' and 'pool.select' TrafficScript functions accept variables as well as literal strings.

```
Boolean getTrafficscriptVariablePoolUse()
```

getUniversalSessionCacheSize()

Get the maximum number of entries in the universal session cache.

```
Unsigned Integer getUniversalSessionCacheSize()
```

getWebcacheDisk()

Get whether the webcache is stored on disk

```
Boolean getWebcacheDisk()
```

getWebcacheDiskDir()

Get the disk cache location

```
String getWebcacheDiskDir()
```

getWebcacheMaxFileNum()

Get the maximum number of files that can be stored in the web cache

```
Unsigned Integer getWebcacheMaxFileNum()
```

getWebcacheMaxFileSize()

Get the largest size of a cacheable object, relative to the total cache size, e.g. '2%', or as an absolute size in kB (default), MB or GB, e.g. '20MB'.

```
String getWebcacheMaxFileSize()
```

getWebcacheNormalizeQuery()

Get whether the assignment sub-strings in the parameter string are put into alphabetical order.

```
Boolean getWebcacheNormalizeQuery()
```

getWebcacheSize()

Get the maximum size of the HTTP web page cache, (specified as a percentage of system RAM, e.g. '20%', or an absolute size, e.g. 200MB)

```
String getWebcacheSize()
```

getWebcacheVerbose()

Get whether an X-Cache-Info header to show cacheability should be added.

```
Boolean getWebcacheVerbose()
```

removeFlipperFrontendCheckAddresses(values) throws InvalidInput, DeploymentError

Remove IP addresses from the list that should be used to check front-end connectivity

```
void removeFlipperFrontendCheckAddresses(
   String[] values
)
```

setASPSessionCacheSize(value) throws InvalidInput, DeploymentError

Set the maximum number of entries in the ASP session cache.

```
void setASPSessionCacheSize(
   Unsigned Integer value
)
```

setAcceptingDelay(value) throws InvalidInput, DeploymentError

Set how often each traffic manager child process checks whether it should be accepting new connections.

```
void setAcceptingDelay(
   Unsigned Integer value
)
```

setAlertEmailInterval(value) throws InvalidInput, DeploymentError

Set the length of time between alert emails, in seconds. Several alert messages will be stored up and sent in one email.

```
void setAlertEmailInterval(
   Unsigned Integer value
)
```

setAlertEmailMaxAttempts(value) throws InvalidInput, DeploymentError

Set the number of times to attempt sending an email before giving up.

```
void setAlertEmailMaxAttempts(
   Unsigned Integer value
)
```

setBackendKeepaliveTimeout(value) throws InvalidInput, DeploymentError

setBackendKeepaliveTimeout is deprecated, please use setIdleConnectionTimeout instead.

```
void setBackendKeepaliveTimeout(
   Unsigned Integer value
```

```
)
```

setBandwidthSharing(value) throws InvalidInput, DeploymentError

This method is now obsolete and is replaced by Catalog.Bandwidth.setSharing.

```
void setBandwidthSharing(
   Boolean value
)
```

setChunkSize(value) throws InvalidInput, DeploymentError

Set the default chunk size for reading and writing data, in bytes.

```
void setChunkSize(
   Unsigned Integer value
)
```

setClientFirstOpt(value) throws InvalidInput, DeploymentError

Set whether client-first network socket optimisations should be used.

```
void setClientFirstOpt(
   Boolean value
)
```

setControlAllowHosts(value) throws InvalidInput, DeploymentError

Set the hosts that are allowed to contact the internal administration port on each traffic manager.

```
void setControlAllowHosts(
   String value
)
```

setDNSCacheExpiryTime(value) throws InvalidInput, DeploymentError

Set the time entries are stored in the DNS cache for, in seconds.

```
void setDNSCacheExpiryTime(
   Unsigned Integer value
)
```

setDNSCacheNegativeExpiryTime(value) throws InvalidInput, DeploymentError

Set the time failed lookups are stored in the DNS cache for, in seconds.

```
void setDNSCacheNegativeExpiryTime(
   Unsigned Integer value
)
```

setDNSCacheSize(value) throws InvalidInput, DeploymentError

Set the maximum number of entries in the DNS cache.

```
void setDNSCacheSize(
   Unsigned Integer value
)
```

setDNSTimeout(value) throws InvalidInput, DeploymentError

Set the timeout for receiving a response from a DNS Server, in seconds.

```
void setDNSTimeout(
   Unsigned Integer value
)
```

setDeadTime(value) throws InvalidInput, DeploymentError

This method is now obsolete and is replaced by Pool.setNodeFailTime.

```
void setDeadTime(
   Unsigned Integer value
)
```

setEC2AccessKeyID(value) throws InvalidInput, DeploymentError

Set the Access Key ID used for interacting with the EC2 API.

```
void setEC2AccessKeyID(
   String value
)
```

setEC2SecretAccessKey(value) throws InvalidInput, DeploymentError

Set the Secret Access Key used for interacting with the EC2 API.

```
void setEC2SecretAccessKey(
   String value
)
```

setErrorLevel(value) throws InvalidInput, DeploymentError

Set the minimum severity of events that should be logged to disk.

```
void setErrorLevel(
```

```
   GlobalSettings.ErrorLevel value
)
```

setErrorLogFile(value) throws InvalidInput, DeploymentError

Set the filename that errors are logged to.

```
void setErrorLogFile(
   String value
)
```

setFTPDataBindLow(value) throws InvalidInput, DeploymentError

Set whether your traffic manager should permit use of FTP data connection source ports lower than 1024. If 'No' your traffic manager can completely drop root privileges, if 'Yes' some or all privileges may be retained in order to bind to low ports.

```
void setFTPDataBindLow(
   Boolean value
)
```

setFlipperArpCount(value) throws InvalidInput, DeploymentError

Set the number of ARP packets each traffic manager sends when an interface is raised.

```
void setFlipperArpCount(
   Unsigned Integer value
)
```

setFlipperAutofailback(value) throws InvalidInput, DeploymentError

Set whether Traffic IPs should automatically failback to recovered machines.

```
void setFlipperAutofailback(
   Boolean value
)
```

setFlipperFrontendCheckAddresses(values) throws InvalidInput, DeploymentError

Set the IP addresses that should be used to check front-end connectivity.

```
void setFlipperFrontendCheckAddresses(
   String[] values
)
```

setFlipperHeartbeatMethod(value) throws InvalidInput, DeploymentError

Set the method used to exchange cluster heartbeat messages.

```
void setFlipperHeartbeatMethod(
   GlobalSettings.FlipperHeartbeatMethod value
)
```

setFlipperMonitorInterval(value) throws InvalidInput, DeploymentError

Set how frequently (in milliseconds) each traffic manager checks and announces its connectivity.

```
void setFlipperMonitorInterval(
   Unsigned Integer value
)
```

setFlipperMonitorTimeout(value) throws InvalidInput, DeploymentError

Set how long (in seconds) each traffic manager waits for a response from its connectivity tests or from other traffic managers before registering a failure.

```
void setFlipperMonitorTimeout(
   Unsigned Integer value
)
```

setFlipperMulticastAddress(value) throws InvalidInput, DeploymentError

Set the multicast address and port used to announce connectivity (e.g. 239.100.1.1:9090).

```
void setFlipperMulticastAddress(
   String value
)
```

setFlipperUnicastPort(value) throws InvalidInput, DeploymentError

Set the unicast UDP port used to announce connectivity (e.g. 9090)

```
void setFlipperUnicastPort(
   Unsigned Integer value
)
```

setFlipperUseBindip(value) throws InvalidInput, DeploymentError

Set whether the heartbeat messages used for fault tolerance are only sent over the management network.

```
void setFlipperUseBindip(
```

```
   Boolean value
)
```

setFlipperVerbose(value) throws InvalidInput, DeploymentError

Set whether the traffic manager should logs all the connectivity tests.

```
void setFlipperVerbose(
   Boolean value
)
```

setHistoricalTrafficDays(value) throws InvalidInput, DeploymentError

Set the length of time historical traffic information is kept for, in days (0=keep indefinitely).

```
void setHistoricalTrafficDays(
   Unsigned Integer value
)
```

setIPSessionCacheSize(value) throws InvalidInput, DeploymentError

Set the maximum number of entries in the IP session cache.

```
void setIPSessionCacheSize(
   Unsigned Integer value
)
```

setIdleConnectionTimeout(value) throws InvalidInput, DeploymentError

Set how long unused HTTP keepalive connections should be kept before being discarded, in seconds.

```
void setIdleConnectionTimeout(
   Unsigned Integer value
)
```

setJ2EESessionCacheSize(value) throws InvalidInput, DeploymentError

Set the maximum number of entries in the J2EE session cache.

```
void setJ2EESessionCacheSize(
   Unsigned Integer value
)
```

setJavaClasspath(value) throws InvalidInput, DeploymentError

Set extra Java CLASSPATH settings required for servlets.

```
void setJavaClasspath(
   String value
)
```

setJavaCommand(value) throws InvalidInput, DeploymentError

Set the command (and arguments) used to start Java.

```
void setJavaCommand(
   String value
)
```

setJavaEnabled(value) throws InvalidInput, DeploymentError

Set whether to enable Java support.

```
void setJavaEnabled(
   Boolean value
)
```

setJavaLib(value) throws InvalidInput, DeploymentError

Set the location of the java library directory

```
void setJavaLib(
   String value
)
```

setJavaMaxConns(value) throws InvalidInput, DeploymentError

Set the maximum number of Java threads

```
void setJavaMaxConns(
   Unsigned Integer value
)
```

setJavaSessionAge(value) throws InvalidInput, DeploymentError

Set the default maximum age of Java session persistence

```
void setJavaSessionAge(
   Unsigned Integer value
)
```

setListenQueueSize(value) throws InvalidInput, DeploymentError

Set the size of the listen queue for managing incoming connections.

```
void setListenQueueSize(
```

```
   Unsigned Integer value
)
```

setLogFlushFlushTime(value) throws InvalidInput, DeploymentError

Set the length of time to wait before flushing the request log files for each virtual server, in seconds.

```
void setLogFlushFlushTime(
   Unsigned Integer value
)
```

setLogInterval(value) throws InvalidInput, DeploymentError

Set the length of time between log messages for log intensive features e.g. SLM, in seconds.

```
void setLogInterval(
   Unsigned Integer value
)
```

setLogRate(value) throws InvalidInput, DeploymentError

Set is the maximum number of connection errors logged per second.

```
void setLogRate(
   Unsigned Integer value
)
```

setLogReopenTime(value) throws InvalidInput, DeploymentError

Set the length of time to wait before re-opening request log files, to handle log file rotation, in seconds.

```
void setLogReopenTime(
   Unsigned Integer value
)
```

setMaxAccepting(value) throws InvalidInput, DeploymentError

Set how many traffic manager child processes accept new connections.

```
void setMaxAccepting(
   Unsigned Integer value
)
```

setMaxIdleConnections(value) throws InvalidInput, DeploymentError

Set the maximum number of unused HTTP keepalive connections to all nodes that should maintained for re-use.

```
void setMaxIdleConnections(
   Unsigned Integer value
)
```

setMaxKeepalives(value) throws InvalidInput, DeploymentError

setMaxKeepalives is deprecated, please use setMaxIdleConnections instead.

```
void setMaxKeepalives(
   Unsigned Integer value
)
```

setMaxRetries(value) throws InvalidInput, DeploymentError

This method is now obsolete and is replaced by Pool.setNodeConnectionAttempts.

```
void setMaxRetries(
   Unsigned Integer value
)
```

setMaximumFDCount(value) throws InvalidInput, DeploymentError

Set the maximum number of file descriptors that your traffic manager will allocate

```
void setMaximumFDCount(
   Unsigned Integer value
)
```

setMonitorNumNodes(value) throws InvalidInput, DeploymentError

Set the maximum number of nodes that can be monitored.

```
void setMonitorNumNodes(
   Unsigned Integer value
)
```

setMultipleAccept(value) throws InvalidInput, DeploymentError

Set whether your traffic manager should try and read multiple new connections each time a new client connects.

```
void setMultipleAccept(
   Boolean value
)
```

setNodeConnectionAttempts(value) throws InvalidInput, DeploymentError

This method is now obsolete and is replaced by Pool.setNodeConnectionAttempts.

```
void setNodeConnectionAttempts(
   Unsigned Integer value
)
```

setNodeFailTime(value) throws InvalidInput, DeploymentError

This method is now obsolete and is replaced by Pool.setNodeFailTime.

```
void setNodeFailTime(
   Unsigned Integer value
)
```

setRateClassLimit(value) throws InvalidInput, DeploymentError

Set the maximum number of Rate classes allowed.

```
void setRateClassLimit(
   Unsigned Integer value
)
```

setRecentConns(value) throws InvalidInput, DeploymentError

Set how many recent connections each traffic manager process should record for the 'active connections' list.

```
void setRecentConns(
   Unsigned Integer value
)
```

setSLMClassLimit(value) throws InvalidInput, DeploymentError

Set the maximum number of SLM classes allowed.

```
void setSLMClassLimit(
   Unsigned Integer value
)
```

setSNMPUserCounters(value) throws InvalidInput, DeploymentError

Set the number of user defined SNMP counters.

```
void setSNMPUserCounters(
   Unsigned Integer value
)
```

setSSL3Ciphers(value) throws InvalidInput, DeploymentError

Set the list of configured SSL ciphers (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s).

```
void setSSL3Ciphers(
   String value
)
```

setSSLDFailureCount(value) throws InvalidInput, DeploymentError

setSSLDFailureCount is deprecated, please use setSSLHardwareFailureCount instead.

```
void setSSLDFailureCount(
   Unsigned Integer value
)
```

setSSLDPKCS11Lib(value) throws InvalidInput, DeploymentError

setSSLDPKCS11Lib is deprecated, please use setSSLHardwarePKCS11Lib instead.

```
void setSSLDPKCS11Lib(
   String value
)
```

setSSLDPKCS11UserPIN(value) throws InvalidInput, DeploymentError

setSSLDPKCS11UserPIN is deprecated, please use setSSLHardwarePKCS11UserPIN instead.

```
void setSSLDPKCS11UserPIN(
   String value
)
```

setSSLHardwareAccelerator(value) throws InvalidInput, DeploymentError

Set whether your traffic manager should always attempt to use SSL hardware.

```
void setSSLHardwareAccelerator(
   Boolean value
)
```

setSSLHardwareFailureCount(value) throws InvalidInput, DeploymentError

Set the number of consecutive failures from the SSL hardware that will be tolerated before your traffic manager tries to log in again.

```
void setSSLHardwareFailureCount(
   Unsigned Integer value
)
```

setSSLHardwarePKCS11Lib(value) throws InvalidInput, DeploymentError

Set the location of the PKCS#11 library supplied by your hardware vendor.

```
void setSSLHardwarePKCS11Lib(
   String value
)
```

setSSLHardwarePKCS11UserPIN(value) throws InvalidInput, DeploymentError

Set the user PIN for the PKCS token (PKCS#11 devices only)

```
void setSSLHardwarePKCS11UserPIN(
   String value
)
```

setSSLHardwareType(value) throws InvalidInput, DeploymentError

Set the device driver library name.

```
void setSSLHardwareType(
   GlobalSettings.SSLHardwareType value
)
```

setSSLSessionCacheSize(value) throws InvalidInput, DeploymentError

Set the maximum number of entries in the SSL session cache. This is used to provide persistence based on SSL session IDs.

```
void setSSLSessionCacheSize(
   Unsigned Integer value
)
```

setSSLSessionIDCacheExpiryTime(value) throws InvalidInput, DeploymentError

Set the length of time that SSL session IDs are stored, in seconds.

```
void setSSLSessionIDCacheExpiryTime(
   Unsigned Integer value
)
```

setSSLSessionIDCacheSize(value) throws InvalidInput, DeploymentError

Set the number of entries in the SSL session ID cache.

```
void setSSLSessionIDCacheSize(
   Unsigned Integer value
)
```

setSSLSupportSSL2(value) throws InvalidInput, DeploymentError

Set whether SSLv2 support is enabled.

```
void setSSLSupportSSL2(
   Boolean value
)
```

setSSLSupportSSL3(value) throws InvalidInput, DeploymentError

Set whether SSLv3 support is enabled.

```
void setSSLSupportSSL3(
   Boolean value
)
```

setSSLSupportTLS1(value) throws InvalidInput, DeploymentError

Set whether TLSv1 support is enabled.

```
void setSSLSupportTLS1(
   Boolean value
)
```

setSSLSupportTLS11(value) throws InvalidInput, DeploymentError

Set whether TLSv1.1 support is enabled.

```
void setSSLSupportTLS11(
   Boolean value
)
```

setSoapIdleMinutes(value) throws InvalidInput, DeploymentError

Set the number of minutes the SOAP server remain idle before exiting

```
void setSoapIdleMinutes(
   Unsigned Integer value
)
```

setSocketOptimizations(value) throws InvalidInput, DeploymentError

Set whether potential network socket optimisations should be used.

```
void setSocketOptimizations(
   GlobalSettings.SocketOptimizations value
)
```

setSsldAccel(value) throws InvalidInput, DeploymentError

setSsldAccel is deprecated, please use setSSLHardwareAccelerator instead.

```
void setSsldAccel(
   Boolean value
)
```

setSsldLibrary(value) throws InvalidInput, DeploymentError

setSsldLibrary is deprecated, please use setSSLHardwareType instead.

```
void setSsldLibrary(
   GlobalSettings.SsldLibrary value
)
```

setStateSyncTime(value) throws InvalidInput, DeploymentError

Set how often the cache state is propagated to other traffic managers in the cluster, in seconds.

```
void setStateSyncTime(
   Unsigned Integer value
)
```

setStateSyncTimeout(value) throws InvalidInput, DeploymentError

Set the timeout for state propagation between cluster members, in seconds

```
void setStateSyncTimeout(
   Unsigned Integer value
)
```

setSystemReadBufferSize(value) throws InvalidInput, DeploymentError

Set the size of the operating system's read buffer, in bytes (0 means use the system default).

```
void setSystemReadBufferSize(
   Unsigned Integer value
)
```

setSystemWriteBufferSize(value) throws InvalidInput, DeploymentError

Set the size of the operating system's write buffer, in bytes (0 means use the system default).

```
void setSystemWriteBufferSize(
   Unsigned Integer value
)
```

setTrafficScriptMemoryWarning(value) throws InvalidInput, DeploymentError

Set the amount of buffered network data a TrafficScript rule can buffer before a warning is logged, in bytes.

```
void setTrafficScriptMemoryWarning(
   Unsigned Integer value
)
```

setTrafficscriptDataSize(value) throws InvalidInput, DeploymentError

Set the maximum size of the TrafficScript shared data pool (specified as a percentage of system RAM, e.g. '5%', or an absolute size, e.g. 200MB)

```
void setTrafficscriptDataSize(
   String value
)
```

setTrafficscriptMaxInstr(value) throws InvalidInput, DeploymentError

Set the maximum number of instructions a TrafficScript rule will run before being aborted.

```
void setTrafficscriptMaxInstr(
   Unsigned Integer value
)
```

setTrafficscriptRegexCacheSize(value) throws InvalidInput, DeploymentError

Set the number of regular expressions to cache

```
void setTrafficscriptRegexCacheSize(
   Unsigned Integer value
)
```

setTrafficscriptVariablePoolUse(value) throws InvalidInput, DeploymentError

Set whether the 'pool.use' and 'pool.select' TrafficScript functions accept variables as well as literal strings.

```
void setTrafficscriptVariablePoolUse(
   Boolean value
)
```

setUniversalSessionCacheSize(value) throws InvalidInput, DeploymentError

Set the maximum number of entries in the universal session cache.

```
void setUniversalSessionCacheSize(
   Unsigned Integer value
)
```

setWebcacheDisk(value) throws InvalidInput, DeploymentError

Set whether the webcache is stored on disk

```
void setWebcacheDisk(
   Boolean value
)
```

setWebcacheDiskDir(value) throws InvalidInput, DeploymentError

Set the disk cache location

```
void setWebcacheDiskDir(
   String value
)
```

setWebcacheMaxFileNum(value) throws InvalidInput, DeploymentError

Set the maximum number of files that can be stored in the web cache

```
void setWebcacheMaxFileNum(
   Unsigned Integer value
)
```

setWebcacheMaxFileSize(value) throws InvalidInput, DeploymentError

Set the largest size of a cacheable object, relative to the total cache size, e.g. '2%', or as an absolute size in kB (default), MB or GB, e.g. '20MB'.

```
void setWebcacheMaxFileSize(
   String value
)
```

setWebcacheNormalizeQuery(value) throws InvalidInput, DeploymentError

Set whether the assignment sub-strings in the parameter string are put into alphabetical order.

```
void setWebcacheNormalizeQuery(
   Boolean value
)
```

setWebcacheSize(value) throws InvalidInput, DeploymentError

Set the maximum size of the HTTP web page cache, (specified as a percentage of system RAM, e.g. '20%', or an absolute size, e.g. 200MB)

```
void setWebcacheSize(
   String value
)
```

setWebcacheVerbose(value) throws InvalidInput, DeploymentError

Set whether an X-Cache-Info header to show cacheability should be added.

```
void setWebcacheVerbose(
   Boolean value
)
```

6.16.2 Enumerations

GlobalSettings.ErrorLevel

```
enum GlobalSettings.ErrorLevel {
   # ERR_FATAL
   fatal,

   # ERR_SERIOUS
   serious,

   # ERR_WARN
   warn,

   # ERR_INFO
   info
}
```

GlobalSettings.FlipperHeartbeatMethod

```
enum GlobalSettings.FlipperHeartbeatMethod {
   # multicast
   multicast,
```

```
   # unicast
   unicast
}
```

GlobalSettings.SSLHardwareType

```
enum GlobalSettings.SSLHardwareType {
   # None
   none,

   # PKCS#11 (e.g. nCipher NetHSM, Sun SCA 6000)
   pkcs11,

   # Cavium Networks CN1000
   cn1000,

   # Cavium Networks CN2000
   cn2000
}
```

GlobalSettings.SocketOptimizations

```
enum GlobalSettings.SocketOptimizations {
   # auto
   auto,

   # Yes
   Yes,

   # No
   No
}
```

GlobalSettings.SsldLibrary

```
enum GlobalSettings.SsldLibrary {
   # None
   none,

   # PKCS#11 (e.g. nCipher NetHSM, Sun SCA 6000)
   pkcs11,

   # Cavium Networks CN1000
   cn1000,

   # Cavium Networks CN2000
   cn2000
}
```

6.17 Conf.Extra

URI: `http://soap.zeus.com/zxtm/1.1/Conf/Extra/`

The Conf.Extra interface allows management of the files stored in the conf/extra directory. These files can be read in by rules, and used as error pages to be sent to clients. This interface allows creating, deleting and retrieving the files.

6.17.1 Methods

deleteFile(names) throws ObjectDoesNotExist

Delete the named files from the conf/extra directory.

```
void deleteFile(
   String[] names
)
```

downloadFile(name) throws ObjectDoesNotExist

Download the named from from the conf/extra directory

```
Binary Data downloadFile(
   String name
)
```

getFileNames()

Get the names of all the files stored in the conf/extra directory.

```
String[] getFileNames()
```

uploadFile(name, content) throws InvalidObjectName

Uploads a new file into the conf/extra, overwriting the file if it already exists.

```
void uploadFile(
   String name
   Binary Data content
)
```

6.18 Diagnose

URI: `http://soap.zeus.com/zxtm/1.1/Diagnose/`

The Diagnose interface provides information about errors and problems in the system.

6.18.1 Methods

activateTrafficManagers(hostnames) throws InvalidInput

Activate traffic managers that have recovered from failures and are ready to start taking Traffic IPs.

```
void activateTrafficManagers(
   String[] hostnames
)
```

diagnoseSystem()

Provides all diagnostic information about the system.

```
Diagnose.ErrorInfo diagnoseSystem()
```

getInactiveTrafficManagers()

List the traffic managers that have recovered from failures and are ready to start taking Traffic IPs.

```
String[] getInactiveTrafficManagers()
```

getTechnicalSupportReport()

Download a technical support report suitable for providing to your support provider to help diagnose problems.

```
Binary Data getTechnicalSupportReport()
```

6.18.2 Structures

Diagnose.AgeError

This structure combines an error message with its age in seconds

```
struct Diagnose.AgeError {
   # Seconds since the error occurred
   Integer age;

   # error message
   String error;
}
```

Diagnose.ConfigError

This structure contains information about configuration errors.

```
struct Diagnose.ConfigError {
   # The file where the error has occurred.
   String filename;

   # The faulty configuration key
   String ConfigKey;

   # Severity of the error
   Diagnose.ErrLevel severity;

   # Date when the error occurred
   Time DetectionDate;

   # A human readable description of the error
   String description;
}
```

Diagnose.ErrorInfo

This structure combines configuration, node, and flipper errors as well as a list of statuses (for an appliance).

```
struct Diagnose.ErrorInfo {
   # The list of traffic managers that could not be contacted.
   String[] NotReachableTrafficManagers;

   # The list of configuration errors.
   Diagnose.ConfigError[] ConfigErrors;

   # The list of flipper errors.
   Diagnose.FlipperError[] FlipperErrors;

   # The list of failed nodes.
   Diagnose.FailedNode[] FailedNodes;

   # The list of system status values.
   Diagnose.SystemStatus[] SystemStatuses;
}
```

Diagnose.FailedNode

This structure contains information about Flipper errors.

```
struct Diagnose.FailedNode {
   # The name of the node that has failed.
   String node;

   # IP address in standard IPv4 or IPv6 notation.
   String IPAddress;
```

```
   # The port number of the node that has failed.
   Integer port;

   # The pool in which this node exists.
   String pool;

   # Time that the failure first occurred.
   Time InitialFailureTime;

   # The last time an attempt was made to connect to the node.
   Time LastConnectionAttempt;

   # The last received error message.
   String ErrorMessage;
}
```

Diagnose.FlipperError

This structure contains information about Flipper errors.

```
struct Diagnose.FlipperError {
   # The name of the affected machine.
   String machine;

   # IP address in standard IPv4 or IPv6 notation.
   String IPAddress;

   # All error messages for that machine.
   Diagnose.AgeError[] errors;
}
```

Diagnose.SystemStatus

Status information about the hardware in an appliance is reported by instances of this structure.

```
struct Diagnose.SystemStatus {
   # The component this object refers to
   String component;

   # The severity level
   Diagnose.ErrLevel severity;

   # Human-readable description of the status
   String message;
}
```

6.18.3 Enumerations

Diagnose.ErrLevel

This enumeration defines the possible severity levels of an error.

```
enum Diagnose.ErrLevel {
   # A fatal error, causes program to die/crash/fail to startup.
   ERR_FATAL,

   # A serious, unexpected error that shouldn't occur under
   # normal conditions. Conditions which will prevent the server
   # from operating properly and should be brought to the
   # webmaster's attention immediately
   ERR_SERIOUS,

   # something which should be brought to the attention of the
   # webmaster, but not immediately.
   ERR_WARN,

   # Minor things that might be of interest e.g. access denied.
   ERR_INFO
}
```

6.19 System.Backups

URI: `http://soap.zeus.com/zxtm/1.0/System/Backups/`

The Backups interfaces provide operations on saved configuration backup archives.

6.19.1 Methods

createBackup(name, description) throws ObjectAlreadyExists, InvalidObjectName

Create backup archive based on the current configuration

```
void createBackup(
   String name
   String description
)
```

deleteAllBackups()

Delete all the backups

```
void deleteAllBackups()
```

deleteBackups(names) throws ObjectDoesNotExist

Delete one or more backups

```
void deleteBackups(
   String[] names
)
```

downloadBackup(name) throws ObjectDoesNotExist

Download a named backup archive

```
Binary Data downloadBackup(
   String name
)
```

getBackupDetails(names)

Get details for one or more backups.

```
System.Backups.Backup[] getBackupDetails(
   String[] names
)
```

listAllBackups()

List the details for all backup archives.

```
System.Backups.Backup[] listAllBackups()
```

restoreBackup(name) throws ObjectDoesNotExist

Restore the named backup archive to be the current configuration

```
void restoreBackup(
   String name
)
```

uploadBackup(name, backup) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput

Upload a backup archive

```
void uploadBackup(
   String name
   Binary Data backup
)
```

6.19.2 Structures

System.Backups.Backup

This structure contains the information for each configuration backup archive.

```
struct System.Backups.Backup {
   # The backup filename.
   String name;

   # The description for this backup.
   String description;

   # The date this backup was created.
   Time date;

   # The version of this backup archive.
   String version;
}
```

6.20 Alerting.EventType

URI: `http://soap.zeus.com/zxtm/1.0/Alerting/EventType/`

Alerting.EventType is an interface that allows you to manage event types. Event Types are groups of events and are associated with a list of actions that are invoked when one of the events in the Event Type is triggered.

6.20.1 Methods

addCustomEvents(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Adds custom events the specified event types will trigger on. Custom events are generated by TrafficScript using the event.emit function. To match all custom events, include '*' in the passed array.

```
void addCustomEvents(
   String[] names
   String[][] events
)
```

addEventType(names, eventtypes) throws ObjectAlreadyExists, InvalidObjectName, InvalidInput, DeploymentError

Add an event type that will cause an action to be triggered when its conditions are met.

```
void addEventType(
   String[] names
```

```
    Alerting.EventType.EventType[] eventtypes
)
```

addEvents(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Adds events to an event type. An event is something that must occur for the associated actions to be triggered (only one event needs to happen to trigger the actions). At least one event must be specified.

```
void addEvents(
    String[] names
    Alerting.EventType.Event[][] events
)
```

addLicensekeyNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of License Key that will trigger the specified event types. If the event type has no License Key names configured, all objects of this type will match.

```
void addLicensekeyNames(
    String[] names
    String[][] objects
)
```

addMappedActions(names, values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add an action that will be run when this event type is triggered.

```
void addMappedActions(
    String[] names
    String[][] values
)
```

addMonitorNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Monitor that will trigger the specified event types. If the event type has no Monitor names configured, all objects of this type will match.

```
void addMonitorNames(
    String[] names
    String[][] objects
)
```

addNodeNames(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Node that will trigger the specified event types. If the event type has no Node names configured, all objects of this type will match.

```
void addNodeNames(
   String[] names
   String[][] events
)
```

addPoolNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Pool that will trigger the specified event types. If the event type has no Pool names configured, all objects of this type will match.

```
void addPoolNames(
   String[] names
   String[][] objects
)
```

addProtectionNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Service Protection Class that will trigger the specified event types. If the event type has no Service Protection Class names configured, all objects of this type will match.

```
void addProtectionNames(
   String[] names
   String[][] objects
)
```

addRuleNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Rule that will trigger the specified event types. If the event type has no Rule names configured, all objects of this type will match.

```
void addRuleNames(
   String[] names
   String[][] objects
)
```

addSlmNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of SLM Class that will trigger the specified event types. If the event type has no SLM Class names configured, all objects of this type will match.

```
void addSlmNames(
   String[] names
   String[][] objects
)
```

addVserverNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Virtual Server that will trigger the specified event types. If the event type has no Virtual Server names configured, all objects of this type will match.

```
void addVserverNames(
   String[] names
   String[][] objects
)
```

addZxtmNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Traffic Manager that will trigger the specified event types. If the event type has no Traffic Manager names configured, all objects of this type will match.

```
void addZxtmNames(
   String[] names
   String[][] objects
)
```

copyEventType(names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError

Copy each of the named event types.

```
void copyEventType(
   String[] names
   String[] new_names
)
```

deleteEventType(names) throws ObjectDoesNotExist, DeploymentError

Removes one or more event types.

```
void deleteEventType(
   String[] names
```

```
)
```

getCustomEvents(names) throws ObjectDoesNotExist

Gets the custom events of the specified event types. Custom events are generated by TrafficScript using the event.emit function. If '*' is returned, all custom events will trigger this event type.

```
String[][] getCustomEvents(
   String[] names
)
```

getEventType(names) throws ObjectDoesNotExist

Returns a set of event type objects for the specified names.

```
Alerting.EventType.EventType[] getEventType(
   String[] names
)
```

getEventTypeNames()

Returns the names of all event types in the system.

```
String[] getEventTypeNames()
```

getEvents(names) throws ObjectDoesNotExist

Gets an event type's events. An event is something that must occur for the associated actions to be triggered (only one event needs to happen to trigger the actions). At least one event must be specified.

```
Alerting.EventType.Event[][] getEvents(
   String[] names
)
```

getLicensekeyNames(names) throws ObjectDoesNotExist

Get the names of License Key that will trigger the specified event types. If the event type has no License Key names configured, all objects of this type will match.

```
String[][] getLicensekeyNames(
   String[] names
)
```

getMappedActions(names) throws ObjectDoesNotExist

Get an action that will be run when this event type is triggered.

```
String[][] getMappedActions(
   String[] names
)
```

getMonitorNames(names) throws ObjectDoesNotExist

Get the names of Monitor that will trigger the specified event types. If the event type has no Monitor names configured, all objects of this type will match.

```
String[][] getMonitorNames(
   String[] names
)
```

getNodeNames(names) throws ObjectDoesNotExist

Get the names of Node that will trigger the specified event types. If the event type has no Node names configured, all objects of this type will match.

```
String[][] getNodeNames(
   String[] names
)
```

getNote(names) throws ObjectDoesNotExist

Get the note for each of the named Event Types.

```
String[] getNote(
   String[] names
)
```

getPoolNames(names) throws ObjectDoesNotExist

Get the names of Pool that will trigger the specified event types. If the event type has no Pool names configured, all objects of this type will match.

```
String[][] getPoolNames(
   String[] names
)
```

getProtectionNames(names) throws ObjectDoesNotExist

Get the names of Service Protection Class that will trigger the specified event types. If the event type has no Service Protection Class names configured, all objects of this type will match.

```
String[][] getProtectionNames(
   String[] names
)
```

getRuleNames(names) throws ObjectDoesNotExist

Get the names of Rule that will trigger the specified event types. If the event type has no Rule names configured, all objects of this type will match.

```
String[][] getRuleNames(
   String[] names
)
```

getSlmNames(names) throws ObjectDoesNotExist

Get the names of SLM Class that will trigger the specified event types. If the event type has no SLM Class names configured, all objects of this type will match.

```
String[][] getSlmNames(
   String[] names
)
```

getVserverNames(names) throws ObjectDoesNotExist

Get the names of Virtual Server that will trigger the specified event types. If the event type has no Virtual Server names configured, all objects of this type will match.

```
String[][] getVserverNames(
   String[] names
)
```

getZxtmNames(names) throws ObjectDoesNotExist

Get the names of Traffic Manager that will trigger the specified event types. If the event type has no Traffic Manager names configured, all objects of this type will match.

```
String[][] getZxtmNames(
   String[] names
)
```

removeCustomEvents(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Removes custom events from the specified event types. Custom events are generated by TrafficScript using the event.emit function. If you pass '*', all custom events will be removed.

```
void removeCustomEvents(
   String[] names
   String[][] events
)
```

removeEvents(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Removes events from the event type. An event is something that must occur for the associated actions to be triggered (only one event needs to happen to trigger the actions). At least one event must be specified.

```
void removeEvents(
   String[] names
   Alerting.EventType.Event[][] events
)
```

removeLicensekeyNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of License Key that will trigger the specified event types. If the event type has no License Key names configured, all objects of this type will match.

```
void removeLicensekeyNames(
   String[] names
   String[][] objects
)
```

removeMappedActions(names, values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove an action that will be run when this event type is triggered.

```
void removeMappedActions(
   String[] names
   String[][] values
)
```

removeMonitorNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Monitor that will trigger the specified event types. If the event type has no Monitor names configured, all objects of this type will match.

```
void removeMonitorNames(
   String[] names
   String[][] objects
)
```

removeNodeNames(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Node that will trigger the specified event types. If the event type has no Node names configured, all objects of this type will match.

```
void removeNodeNames(
   String[] names
   String[][] events
)
```

removePoolNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Pool that will trigger the specified event types. If the event type has no Pool names configured, all objects of this type will match.

```
void removePoolNames(
   String[] names
   String[][] objects
)
```

removeProtectionNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Service Protection Class that will trigger the specified event types. If the event type has no Service Protection Class names configured, all objects of this type will match.

```
void removeProtectionNames(
   String[] names
   String[][] objects
)
```

removeRuleNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Rule that will trigger the specified event types. If the event type has no Rule names configured, all objects of this type will match.

```
void removeRuleNames(
   String[] names
   String[][] objects
)
```

removeSlmNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of SLM Class that will trigger the specified event types. If the event type has no SLM Class names configured, all objects of this type will match.

```
void removeSlmNames(
   String[] names
   String[][] objects
)
```

removeVserverNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Virtual Server that will trigger the specified event types. If the event type has no Virtual Server names configured, all objects of this type will match.

```
void removeVserverNames(
   String[] names
   String[][] objects
)
```

removeZxtmNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Traffic Manager that will trigger the specified event types. If the event type has no Traffic Manager names configured, all objects of this type will match.

```
void removeZxtmNames(
   String[] names
   String[][] objects
)
```

renameEventType(names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError, InvalidOperation

Rename each of the named event types.

```
void renameEventType(
   String[] names
   String[] new_names
)
```

setCustomEvents(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Gets the custom events the specified event types will trigger on. Custom events are generated by TrafficScript using the event.emit function. To match all custom events, include '*' in the passed array.

```
void setCustomEvents(
   String[] names
   String[][] events
)
```

setEvents(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Sets an event type's events (all old events will be removed). An event is something that must occur for the associated actions to be triggered (only one event needs to happen to trigger the actions). At least one event must be specified.

```
void setEvents(
   String[] names
   Alerting.EventType.Event[][] events
)
```

setLicensekeyNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of License Key that will trigger the specified event types. If the event type has no License Key names configured, all objects of this type will match.

```
void setLicensekeyNames(
   String[] names
   String[][] objects
)
```

setMappedActions(names, values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set an action that will be run when this event type is triggered.

```
void setMappedActions(
   String[] names
   String[][] values
)
```

setMonitorNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Monitor that will trigger the specified event types. If the event type has no Monitor names configured, all objects of this type will match.

```
void setMonitorNames(
   String[] names
   String[][] objects
)
```

setNodeNames(names, events) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Node that will trigger the specified event types. If the event type has no Node names configured, all objects of this type will match.

```
void setNodeNames(
   String[] names
   String[][] events
)
```

setNote(names, values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the note for each of the named Event Types.

```
void setNote(
   String[] names
   String[] values
)
```

setPoolNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Pool that will trigger the specified event types. If the event type has no Pool names configured, all objects of this type will match.

```
void setPoolNames(
   String[] names
   String[][] objects
)
```

setProtectionNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Service Protection Class that will trigger the specified event types. If the event type has no Service Protection Class names configured, all objects of this type will match.

```
void setProtectionNames(
   String[] names
   String[][] objects
)
```

setRuleNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Rule that will trigger the specified event types. If the event type has no Rule names configured, all objects of this type will match.

```
void setRuleNames(
   String[] names
   String[][] objects
)
```

setSlmNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of SLM Class that will trigger the specified event types. If the event type has no SLM Class names configured, all objects of this type will match.

```
void setSlmNames(
   String[] names
   String[][] objects
)
```

setVserverNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Virtual Server that will trigger the specified event types. If the event type has no Virtual Server names configured, all objects of this type will match.

```
void setVserverNames(
   String[] names
   String[][] objects
)
```

setZxtmNames(names, objects) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of Traffic Manager that will trigger the specified event types. If the event type has no Traffic Manager names configured, all objects of this type will match.

```
void setZxtmNames(
   String[] names
   String[][] objects
)
```

6.20.2 Structures

Alerting.EventType.EventType

A set of conditions that when met causes an action to be run.

```
struct Alerting.EventType.EventType {
   # The events that will trigger the associated actions.
   Alerting.EventType.Event[] events;

   # The names of all the custom events you want to trigger this
   # event type.
   String[] customEvents;

   # The names of all the actions mapped to this custom event.
   String[] mappedActions;

   # The names of all the Service Protection Classes that should
   # trigger this event type. If this is an empty array all
   # objects of this type will be matched.
   String[] protectionNames;

   # The names of all the Virtual Servers that should trigger
   # this event type. If this is an empty array all objects of
   # this type will be matched.
   String[] vserverNames;

   # The names of all the SLM Classes that should trigger this
   # event type. If this is an empty array all objects of this
   # type will be matched.
   String[] slmNames;

   # The names of all the Monitors that should trigger this event
   # type. If this is an empty array all objects of this type
   # will be matched.
   String[] monitorNames;

   # The names of all the Rules that should trigger this event
   # type. If this is an empty array all objects of this type
   # will be matched.
```

```
    String[] ruleNames;

    # The names of all the License Keys that should trigger this
    # event type. If this is an empty array all objects of this
    # type will be matched.
    String[] licensekeyNames;

    # The names of all the Traffic Managers that should trigger
    # this event type. If this is an empty array all objects of
    # this type will be matched.
    String[] zxtmNames;

    # The names of all the Pools that should trigger this event
    # type. If this is an empty array all objects of this type
    # will be matched.
    String[] poolNames;
}
```

6.20.3 Enumerations

Alerting.EventType.Event

```
enum Alerting.EventType.Event {
    # This event matches all events.
    ALL,

    # Special value that matches all events of type Configuration
    # Files.
    config_ALL,

    # Configuration Files - Configuration file added
    config_confadd,

    # Configuration Files - Configuration file deleted
    config_confdel,

    # Configuration Files - Configuration file modified
    config_confmod,

    # Configuration Files - Configuration file now OK
    config_confok,

    # Special value that matches all events of type Fault
    # Tolerance.
    faulttolerance_ALL,

    # Fault Tolerance - Activating this machine automatically
    # because it is the only working machine in its Traffic IP
    # Groups
    faulttolerance_activatealldead,
```

```
# Fault Tolerance - Machine has recovered and been activated
# automatically because it would cause no service disruption
faulttolerance_activatedautomatically,

# Fault Tolerance - All machines are working
faulttolerance_allmachinesok,

# Fault Tolerance - Removing EC2 Elastic IP Address from all
# machines; it is no longer a part of any Traffic IP Groups.
faulttolerance_dropec2ipwarn,

# Fault Tolerance - Dropping Traffic IP Address due to a
# configuration change or traffic manager recovery
faulttolerance_dropipinfo,

# Fault Tolerance - Dropping Traffic IP Address due to an
# error
faulttolerance_dropipwarn,

# Fault Tolerance - Moving EC2 Elastic IP Address; local
# machine is working
faulttolerance_ec2flipperraiselocalworking,

# Fault Tolerance - Moving EC2 Elastic IP Address; other
# machines have failed
faulttolerance_ec2flipperraiseothersdead,

# Fault Tolerance - Problem occurred when managing an Elastic
# IP address
faulttolerance_ec2iperr,

# Fault Tolerance - Cannot raise Elastic IP on this machine
# until EC2 provides it with a public IP address.
faulttolerance_ec2nopublicip,

# Fault Tolerance - Back-end nodes are now working
faulttolerance_flipperbackendsworking,

# Fault Tolerance - Re-raising Traffic IP Address; Operating
# system did not fully raise the address
faulttolerance_flipperdadreraise,

# Fault Tolerance - Frontend machines are now working
faulttolerance_flipperfrontendsworking,

# Fault Tolerance - Failed to raise Traffic IP Address; the
# address exists elsewhere on your network and cannot be
# raised
faulttolerance_flipperipexists,
```

```
# Fault Tolerance - Raising Traffic IP Address; local machine
# is working
faulttolerance_flipperraiselocalworking,

# Fault Tolerance - Raising Traffic IP Address; Operating
# System had dropped this IP address
faulttolerance_flipperraiseosdrop,

# Fault Tolerance - Raising Traffic IP Address; other machines
# have failed
faulttolerance_flipperraiseothersdead,

# Fault Tolerance - Machine is ready to raise Traffic IP
# addresses
faulttolerance_flipperrecovered,

# Fault Tolerance - Remote machine has failed
faulttolerance_machinefail,

# Fault Tolerance - Remote machine is now working
faulttolerance_machineok,

# Fault Tolerance - Remote machine has recovered and can raise
# Traffic IP addresses
faulttolerance_machinerecovered,

# Fault Tolerance - Remote machine has timed out and been
# marked as failed
faulttolerance_machinetimeout,

# Fault Tolerance - The amount of load handled by the local
# machine destined for this Traffic IP has changed.
faulttolerance_multihostload,

# Fault Tolerance - Failed to ping back-end nodes
faulttolerance_pingbackendfail,

# Fault Tolerance - Failed to ping any of the machines used to
# check the front-end connectivity
faulttolerance_pingfrontendfail,

# Fault Tolerance - Failed to ping default gateway
faulttolerance_pinggwfail,

# Fault Tolerance - Received an invalid response from another
# cluster member
faulttolerance_statebaddata,

# Fault Tolerance - Failed to connect to another cluster
# member for state sharing
faulttolerance_stateconnfail,
```

```
# Fault Tolerance - Successfully connected to another cluster
# member for state sharing
faulttolerance_stateok,

# Fault Tolerance - Reading state data from another cluster
# member failed
faulttolerance_statereadfail,

# Fault Tolerance - Timeout while sending state data to
# another cluster member
faulttolerance_statetimeout,

# Fault Tolerance - Received unexpected state data from
# another cluster member
faulttolerance_stateunexpected,

# Fault Tolerance - Writing state data to another cluster
# member failed
faulttolerance_statewritefail,

# Fault Tolerance - An error occurred when using the zcluster
# Multi-Hosted IP kernel module
faulttolerance_zclustermoderr,

# Special value that matches all events of type General.
general_ALL,

# General - An error occurred during user authentication
general_autherror,

# General - Running out of free file descriptors
general_fewfreefds,

# General - Failed to load geolocation data.
general_geodataloadfail,

# General - Appliance hardware notification
general_hardware,

# General - Software must be restarted to apply configuration
# changes
general_restartrequired,

# General - Software is running
general_running,

# General - Time has been moved back
general_timemovedback,

# General - Zeus Traffic Manager software problem
```

```
general_zxtmswerror,

# Special value that matches all events of type Java.
java_ALL,

# Java - Java runner died
java_javadied,

# Java - Cannot start Java, program not found
java_javanotfound,

# Java - Java started
java_javastarted,

# Java - Java failed to start
java_javastartfail,

# Java - Java support has stopped
java_javastop,

# Java - Java failed to terminate
java_javaterminatefail,

# Java - Servlet encountered an error
java_servleterror,

# Special value that matches all events of type License Keys.
licensekeys_ALL,

# License Keys - License key bandwidth limit has been hit
licensekeys_bwlimited,

# License Keys - Configured cache size exceeds license limit,
# only using amount allowed by license
licensekeys_cachesizereduced,

# License Keys - License key has expired
licensekeys_expired,

# License Keys - License key expires within 4 days
licensekeys_expiresoon,

# License Keys - License allows less memory for caching
licensekeys_lessmemallowed,

# License Keys - License key authorized
licensekeys_license-authorized,

# License Keys - Unable to authorize license key
licensekeys_license-graceperiodexpired,
```

```
# License Keys - License server rejected license key; key
# remains authorized
licensekeys_license-rejected-authorized,

# License Keys - License server rejected license key; key is
# not authorized
licensekeys_license-rejected-unauthorized,

# License Keys - Unable to contact license server; license key
# remains authorized
licensekeys_license-timedout-authorized,

# License Keys - Unable to contact license server; license key
# is not authorized
licensekeys_license-timedout-unauthorized,

# License Keys - License key is not authorized
licensekeys_license-unauthorized,

# License Keys - Cluster size exceeds license key limit
licensekeys_licenseclustertoobig,

# License Keys - License key is corrupt
licensekeys_licensecorrupt,

# License Keys - License allows more memory for caching
licensekeys_morememallowed,

# License Keys - License key SSL transactions-per-second limit
# has been hit
licensekeys_ssltpslimited,

# License Keys - License key transactions-per-second limit has
# been hit
licensekeys_tpslimited,

# License Keys - Started without a license
licensekeys_unlicensed,

# License Keys - Using a development license
licensekeys_usingdevlicense,

# License Keys - Using license key
licensekeys_usinglicense,

# Special value that matches all events of type Monitors.
monitors_ALL,

# Monitors - Monitor has detected a failure
monitors_monitorfail,
```

```
# Monitors - Monitor is working
monitors_monitorok,

# Special value that matches all events of type Pools.
pools_ALL,

# Pools - HTTP response contained an invalid Content-Length
# header
pools_badcontentlen,

# Pools - Node returned invalid EHLO response
pools_ehloinvalid,

# Pools - Node has failed
pools_nodefail,

# Pools - Failed to resolve node address
pools_noderesolvefailure,

# Pools - Node resolves to multiple IP addresses
pools_noderesolvemultiple,

# Pools - Node is working again
pools_nodeworking,

# Pools - Node doesn't provide STARTTLS support
pools_nostarttls,

# Pools - Pool has no back-end nodes responding
pools_pooldied,

# Pools - Pool configuration contains no valid backend nodes
pools_poolnonodes,

# Pools - Pool now has working nodes
pools_poolok,

# Pools - Node returned invalid STARTTLS response
pools_starttlsinvalid,

# Special value that matches all events of type Service
# Protection Classes.
protection_ALL,

# Service Protection Classes - Summary of recent service
# protection events
protection_triggersummary,

# Special value that matches all events of type Rules.
rules_ALL,
```

```
# Rules - data.set() has run out of space
rules_datastorefull,

# Rules - Rule selected an unresolvable host
rules_forwardproxybadhost,

# Rules - Rule used event.emit() with an invalid custom event
rules_invalidemit,

# Rules - Rule selected an unknown rate shaping class
rules_norate,

# Rules - Rule references an unknown pool via pool.activenodes
rules_poolactivenodesunknown,

# Rules - Rule selected an unknown pool
rules_pooluseunknown,

# Rules - Rule aborted during execution
rules_ruleabort,

# Rules - Rule encountered invalid data while uncompressing
# response
rules_rulebodycomperror,

# Rules - Rule has buffered more data than expected
rules_rulebufferlarge,

# Rules - Rule logged an info message using log.info
rules_rulelogmsginfo,

# Rules - Rule logged an error message using log.error
rules_rulelogmsgserious,

# Rules - Rule logged a warning message using log.warn
rules_rulelogmsgwarn,

# Rules - Rule selected an unknown session persistence class
rules_rulenopersistence,

# Rules - Client sent invalid HTTP request body
rules_rulesinvalidrequestbody,

# Rules - Attempt to use http.getResponse or
# http.getResponseBody after http.stream.startResponse
rules_rulestreamerrorgetresponse,

# Rules - Internal error while processing HTTP stream
rules_rulestreamerrorinternal,

# Rules - Rule did not supply enough data in HTTP stream
```

```
rules_rulestreamerrornotenough,

# Rules - Attempt to initialize HTTP stream before previous
# stream had finished
rules_rulestreamerrornotfinished,

# Rules - Attempt to stream data or finish a stream before
# streaming had been initialized
rules_rulestreamerrornotstarted,

# Rules - Data supplied to HTTP stream could not be processed
rules_rulestreamerrorprocessfailure,

# Rules - Rule supplied too much data in HTTP stream
rules_rulestreamerrortoomuch,

# Rules - Rule encountered an XML error
rules_rulexmlerr,

# Special value that matches all events of type SLM Classes.
slm_ALL,

# SLM Classes - SLM shared class limit exceeded
slm_slmclasslimitexceeded,

# SLM Classes - SLM has fallen below serious threshold
slm_slmfallenbelowserious,

# SLM Classes - SLM has fallen below warning threshold
slm_slmfallenbelowwarn,

# SLM Classes - Node information when SLM is non-conforming
# (no SNMP trap)
slm_slmnodeinfo,

# SLM Classes - SLM has risen above the serious threshold
slm_slmrecoveredserious,

# SLM Classes - SLM has recovered
slm_slmrecoveredwarn,

# Special value that matches all events of type SSL Hardware.
sslhw_ALL,

# SSL Hardware - SSL hardware support failed
sslhw_sslhwfail,

# SSL Hardware - SSL hardware support restarted
sslhw_sslhwrestart,

# SSL Hardware - SSL hardware support started
```

```
sslhw_sslhwstart,

# All custom TrafficScript events.
trafficscript_ALL,

# Special value that matches all events of type Virtual
# Servers.
vservers_ALL,

# Virtual Servers - A protocol error has occurred
vservers_connerror,

# Virtual Servers - A socket connection failure has occurred
vservers_connfail,

# Virtual Servers - A virtual server request log file was
# deleted (Zeus Appliances only)
vservers_logfiledeleted,

# Virtual Servers - Dropped connection, request exceeded
# max_client_buffer limit
vservers_maxclientbufferdrop,

# Virtual Servers - Pool uses a session persistence class that
# does not work with this virtual server's protocol
vservers_poolpersistencemismatch,

# Virtual Servers - Private key now OK
vservers_privkeyok,

# Virtual Servers - Error compressing HTTP response
vservers_respcompfail,

# Virtual Servers - Response from webserver too large
vservers_responsetoolarge,

# Virtual Servers - No suitable ports available for streaming
# data connection
vservers_rtspstreamnoports,

# Virtual Servers - No suitable ports available for streaming
# data connection
vservers_sipstreamnoports,

# Virtual Servers - Request(s) received while SSL
# configuration invalid, connection closed
vservers_ssldrop,

# Virtual Servers - One or more SSL connections from clients
# failed recently
vservers_sslfail,
```

```
    # Virtual Servers - Certificate Authority certificate expired
    vservers_vscacertexpired,

    # Virtual Servers - Certificate Authority certificate will
    # expire soon
    vservers_vscacerttoexpire,

    # Virtual Servers - CRL for a Certificate Authority is out of
    # date
    vservers_vscrloutofdate,

    # Virtual Servers - Failed to write log file for virtual
    # server
    vservers_vslogwritefail,

    # Virtual Servers - Public SSL certificate expired
    vservers_vssslcertexpired,

    # Virtual Servers - Public SSL certificate will expire soon
    vservers_vssslcerttoexpire,

    # Virtual Servers - Virtual server started
    vservers_vsstart,

    # Virtual Servers - Virtual server stopped
    vservers_vsstop,

    # Special value that matches all events of type Traffic
    # Managers.
    zxtms_ALL,

    # Traffic Managers - Configuration update refused: traffic
    # manager version mismatch
    zxtms_versionmismatch
}
```

6.21 Alerting.Action

URI: `http://soap.zeus.com/zxtm/1.0/Alerting/Action/`

Alerting.Action is an interface that allows you to add actions that are run by event types.

6.21.1 Methods

addAction(names, types) throws InvalidInput, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add a action that can be run by an event.

```
void addAction(
   String[] names
   Alerting.Action.Type[] types
)
```

addScriptArguments(names, arguments) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Adds a set of arguments to the specified actions. The actions specified must be of type 'program'.

```
void addScriptArguments(
   String[] names
   Alerting.Action.Argument[][] arguments
)
```

copyAction(names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Copy each of the named actions.

```
void copyAction(
   String[] names
   String[] new_names
)
```

deleteAction(names) throws ObjectDoesNotExist, ObjectInUse, DeploymentError

Deletes each of the named actions.

```
void deleteAction(
   String[] names
)
```

deleteActionProgram(names) throws ObjectDoesNotExist, DeploymentError, ObjectInUse

Delete the named action programs.

```
void deleteActionProgram(
   String[] names
)
```

downloadActionProgram(name) throws ObjectDoesNotExist

Download the named action program.

```
Binary Data downloadActionProgram(
   String name
)
```

getActionNames()

Get the names of all available actions.

```
String[] getActionNames()
```

getActionNamesOfType(type)

Get the names of all actions of the specified type.

```
String[] getActionNamesOfType(
   Alerting.Action.Type type
)
```

getActionProgramNames()

Get the names of all the uploaded action programs. These are the programs that can be executed by custom program actions.

```
String[] getActionProgramNames()
```

getActionType(names) throws InvalidOperation, ObjectDoesNotExist

Returns the type of each of the named actions.

```
Alerting.Action.Type[] getActionType(
   String[] names
)
```

getEmailRecipients(names) throws InvalidOperation, ObjectDoesNotExist

Get the address the alert emails are sent from.

```
String[] getEmailRecipients(
   String[] names
)
```

getEmailSMTPServer(names) throws InvalidOperation, ObjectDoesNotExist

Get the SMTP server used to send alert emails for the specified actions.

```
String[] getEmailSMTPServer(
   String[] names
)
```

getEmailSender(names) throws InvalidOperation, ObjectDoesNotExist

Get the specified email addresses to the recipient list for the specified actions.

```
String[] getEmailSender(
   String[] names
)
```

getLogFilePath(names) throws InvalidOperation, ObjectDoesNotExist

Get the file this action logs to.

```
String[] getLogFilePath(
   String[] names
)
```

getSOAPAdditional(names) throws InvalidOperation, ObjectDoesNotExist

Get the additional information to send with the SOAP alert call.

```
String[] getSOAPAdditional(
   String[] names
)
```

getSOAPAuthentication(names) throws InvalidOperation, ObjectDoesNotExist

Gets the username used to log in with HTTP basic authentication. The actions specified must be of type 'soap'. Note that the password field is always returned as an empty string.

```
Alerting.Action.Login[] getSOAPAuthentication(
   String[] names
)
```

getSOAPProxy(names) throws InvalidOperation, ObjectDoesNotExist

Get the server the SOAP event call will be made to for each of the specified SOAP events.

```
String[] getSOAPProxy(
   String[] names
)
```

getScriptArguments(names) throws InvalidOperation, ObjectDoesNotExist

Gets all arguments for the specified script actions. The actions specified must be of type 'program'.

```
Alerting.Action.Argument[][] getScriptArguments(
   String[] names
```

```
)
```

getScriptProgram(names) throws InvalidOperation, ObjectDoesNotExist

Get the program to run including its command line arguments. The actions specified must be of type 'program'.

```
String[] getScriptProgram(
   String[] names
)
```

getSyslogHost(names) throws InvalidOperation, ObjectDoesNotExist

Get the host to send syslog messages to (if empty, messages will be sent to localhost). The actions specified must be of type 'syslog'.

```
String[] getSyslogHost(
   String[] names
)
```

getTimeout(names) throws InvalidOperation, ObjectDoesNotExist

Get how long an action has to run, in seconds (set to 0 disable timing out).

```
Unsigned Integer[] getTimeout(
   String[] names
)
```

getTrapCommunity(names) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP community string for the SNMP trap. The actions specified must be of type 'trap'.

```
String[] getTrapCommunity(
   String[] names
)
```

getTrapHost(names) throws InvalidOperation, ObjectDoesNotExist

Get the hostname or IPv4 address and optional port number that should receive the SNMP trap. The actions specified must be of type 'trap'.

```
String[] getTrapHost(
   String[] names
)
```

getVerbose(names) throws InvalidOperation, ObjectDoesNotExist

Get if verbose logging is enabled for this action.

```
Boolean[] getVerbose(
   String[] names
)
```

removeSOAPAuthentication(names) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Disables using HTTP basic authentication with the SOAP Call. The actions specified must be of type 'soap'.

```
void removeSOAPAuthentication(
   String[] names
)
```

removeScriptArguments(names, arguments) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Removes a set of arguments from the specified script actions. The actions specified must be of type 'program'.

```
void removeScriptArguments(
   String[] names
   String[][] arguments
)
```

renameAction(names, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename each of the named actions.

```
void renameAction(
   String[] names
   String[] new_names
)
```

setEmailRecipients(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the address the alert emails are sent from.

```
void setEmailRecipients(
   String[] names
   String[] values
)
```

setEmailSMTPServer(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SMTP server used to send alert emails for the specified actions.

```
void setEmailSMTPServer(
   String[] names
   String[] values
)
```

setEmailSender(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the specified email addresses to the recipient list for the specified actions.

```
void setEmailSender(
   String[] names
   String[] values
)
```

setLogFilePath(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the file this action logs to.

```
void setLogFilePath(
   String[] names
   String[] values
)
```

setSOAPAdditional(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the additional information to send with the SOAP alert call.

```
void setSOAPAdditional(
   String[] names
   String[] values
)
```

setSOAPAuthentication(names, credentials) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Sets the username and password to use to log in with HTTP basic authentication. The actions specified must be of type 'soap'.

```
void setSOAPAuthentication(
   String[] names
   Alerting.Action.Login[] credentials
```

```
)
```

setSOAPProxy(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the server the SOAP event call will be made to for each of the specified SOAP events.

```
void setSOAPProxy(
   String[] names
   String[] values
)
```

setScriptProgram(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the program to run including its command line arguments. The actions specified must be of type 'program'.

```
void setScriptProgram(
   String[] names
   String[] values
)
```

setSyslogHost(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the host to send syslog messages to (if empty, messages will be sent to localhost). The actions specified must be of type 'syslog'.

```
void setSyslogHost(
   String[] names
   String[] values
)
```

setTimeout(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set how long an action has to run, in seconds (set to 0 disable timing out).

```
void setTimeout(
   String[] names
   Unsigned Integer[] values
)
```

setTrapCommunity(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SNMP community string for the SNMP trap. The actions specified must be of type 'trap'.

```
void setTrapCommunity(
   String[] names
   String[] values
)
```

setTrapHost(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the hostname or IPv4 address and optional port number that should receive the SNMP trap. The actions specified must be of type 'trap'.

```
void setTrapHost(
   String[] names
   String[] values
)
```

setVerbose(names, values) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set if verbose logging is enabled for this action.

```
void setVerbose(
   String[] names
   Boolean[] values
)
```

testAction(names) throws ObjectDoesNotExist

Sends a test event to the named actions to confirm that they are working as expected.

```
void testAction(
   String[] names
)
```

updateScriptArguments(names, argument_names, new_arguments) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Allows arguments for the the specified script actions to be changed. The actions specified must be of type 'program'.

```
void updateScriptArguments(
   String[] names
   String[][] argument_names
```

```
   Alerting.Action.Argument[][] new_arguments
)
```

uploadActionProgram(name, content) throws InvalidObjectName, DeploymentError

Uploads an action program, overwriting the file if it already exists.

```
void uploadActionProgram(
   String name
   Binary Data content
)
```

6.21.2 Structures

Alerting.Action.Argument

An argument that is added to the command line when the script is run

```
struct Alerting.Action.Argument {
   # The name of the argument.
   String name;

   # The value of the argument.
   String value;

   # A description of the argument.
   String description;
}
```

Alerting.Action.Login

An argument that is added to the command line when the script is run

```
struct Alerting.Action.Login {
   # The username for basic SOAP authentication
   String username;

   # The username for basic SOAP authentication
   String password;
}
```

6.21.3 Enumerations

Alerting.Action.Type

```
enum Alerting.Action.Type {
   # Sends e-mails to a set of e-mail addresses.
```

```
   email,

   # Reports event information to a remote server using the SOAP
   # protocol.
   soap,

   # Sends an SNMP message to a remote server.
   trap,

   # Writes a log line in the syslog.
   syslog,

   # Writes a log line in a named file.
   log,

   # Executes an external program.
   program
}
```

6.22 AlertCallback

URI: `http://soap.zeus.com/zxtm/1.0/AlertCallback/`

AlertCallback is a callback interface that can be implemented on a separate server to receive events via SOAP from the traffic manager. This interface is not implemented by traffic manager itself.

6.22.1 Methods

eventOccurred(zxtm, time, severity, primary_tag, tags, objects, description, additional, event_type)

This function is used by the traffic manager to report an event using a SOAP call. You can easily identify the event being reported using the primary_tag field, which is the event's unique identifier. The tags array is reserved for future use, and will be empty.

```
void eventOccurred(
   String zxtm
   Time time
   AlertCallback.Severity severity
   AlertCallback.Tag primary_tag
   AlertCallback.Tag[] tags
   AlertCallback.Object[] objects
   String description
   String additional
   String event_type
)
```

6.22.2 Structures

AlertCallback.Object

Information on an object that triggered this event.

```
struct AlertCallback.Object {
   # The type of the object
   AlertCallback.ObjectType type;

   # The name of the object.
   String name;
}
```

6.22.3 Enumerations

AlertCallback.ObjectType

```
enum AlertCallback.ObjectType {
   # An unexpected type
   Unknown,

   # Actions
   actions,

   # Authenticators
   auth,

   # Bandwidth Classes
   bandwidth,

   # Configuration Files
   config,

   # DNS Lookup
   dns,

   # Event Types
   events,

   # TrafficScript Resources
   extra,

   # Fault Tolerance
   faulttolerance,

   # Traffic IPs
   flipper,

   # General
```

```
general,

# HTTP Events
http,

# Java Resources
jars,

# Java
java,

# License Keys
licensekeys,

# Monitors
monitors,

# Nodes
nodes,

# Session Persistence Classes
persistence,

# Processes
pids,

# Pools
pools,

# Service Protection Classes
protection,

# Rate Classes
rate,

# RTSP Events
rtsp,

# Rules
rules,

servlet,

# Java Servlets
servlets,

# SIP Events
sip,

# SLM Classes
slm,
```

```
    # SMTP Events
    smtp,

    # SSL Hardware
    sslhw,

    # SIP/RTSP
    streaming,

    # Traffic IPs
    tips,

    # Custom Events
    trafficscript,

    # Virtual Servers
    vservers,

    # Traffic Managers
    zxtms
}
```

AlertCallback.Severity

```
enum AlertCallback.Severity {
    # Denial of Service Event
    DOS,

    # Fatal Error Event
    FATAL,

    # Information Event
    INFO,

    # Serious Error Event
    SERIOUS,

    # SSL Error Event
    SSL,

    # Warning Event
    WARN
}
```

AlertCallback.Tag

```
enum AlertCallback.Tag {
    # This tag is used to with emitting a custom event generated
    # with the TrafficScript function 'event.emit'. Look at the
```

```
# object that came with the callback to see the name of the
# custom event
CustomEvent,

# An unknown tag
Unknown,

# Configuration Files - Configuration file added
config_confadd,

# Configuration Files - Configuration file deleted
config_confdel,

# Configuration Files - Configuration file modified
config_confmod,

# Configuration Files - Configuration file now OK
config_confok,

# Fault Tolerance - Activating this machine automatically
# because it is the only working machine in its Traffic IP
# Groups
faulttolerance_activatealldead,

# Fault Tolerance - Machine has recovered and been activated
# automatically because it would cause no service disruption
faulttolerance_activatedautomatically,

# Fault Tolerance - All machines are working
faulttolerance_allmachinesok,

# Fault Tolerance - Removing EC2 Elastic IP Address from all
# machines; it is no longer a part of any Traffic IP Groups.
faulttolerance_dropec2ipwarn,

# Fault Tolerance - Dropping Traffic IP Address due to a
# configuration change or traffic manager recovery
faulttolerance_dropipinfo,

# Fault Tolerance - Dropping Traffic IP Address due to an
# error
faulttolerance_dropipwarn,

# Fault Tolerance - Moving EC2 Elastic IP Address; local
# machine is working
faulttolerance_ec2flipperraiselocalworking,

# Fault Tolerance - Moving EC2 Elastic IP Address; other
# machines have failed
faulttolerance_ec2flipperraiseothersdead,
```

```
# Fault Tolerance - Problem occurred when managing an Elastic
# IP address
faulttolerance_ec2iperr,

# Fault Tolerance - Cannot raise Elastic IP on this machine
# until EC2 provides it with a public IP address.
faulttolerance_ec2nopublicip,

# Fault Tolerance - Back-end nodes are now working
faulttolerance_flipperbackendsworking,

# Fault Tolerance - Re-raising Traffic IP Address; Operating
# system did not fully raise the address
faulttolerance_flipperdadreraise,

# Fault Tolerance - Frontend machines are now working
faulttolerance_flipperfrontendsworking,

# Fault Tolerance - Failed to raise Traffic IP Address; the
# address exists elsewhere on your network and cannot be
# raised
faulttolerance_flipperipexists,

# Fault Tolerance - Raising Traffic IP Address; local machine
# is working
faulttolerance_flipperraiselocalworking,

# Fault Tolerance - Raising Traffic IP Address; Operating
# System had dropped this IP address
faulttolerance_flipperraiseosdrop,

# Fault Tolerance - Raising Traffic IP Address; other machines
# have failed
faulttolerance_flipperraiseothersdead,

# Fault Tolerance - Machine is ready to raise Traffic IP
# addresses
faulttolerance_flipperrecovered,

# Fault Tolerance - Remote machine has failed
faulttolerance_machinefail,

# Fault Tolerance - Remote machine is now working
faulttolerance_machineok,

# Fault Tolerance - Remote machine has recovered and can raise
# Traffic IP addresses
faulttolerance_machinerecovered,

# Fault Tolerance - Remote machine has timed out and been
# marked as failed
```

```
faulttolerance_machinetimeout,

# Fault Tolerance - The amount of load handled by the local
# machine destined for this Traffic IP has changed.
faulttolerance_multihostload,

# Fault Tolerance - Failed to ping back-end nodes
faulttolerance_pingbackendfail,

# Fault Tolerance - Failed to ping any of the machines used to
# check the front-end connectivity
faulttolerance_pingfrontendfail,

# Fault Tolerance - Failed to ping default gateway
faulttolerance_pinggwfail,

# Fault Tolerance - Received an invalid response from another
# cluster member
faulttolerance_statebaddata,

# Fault Tolerance - Failed to connect to another cluster
# member for state sharing
faulttolerance_stateconnfail,

# Fault Tolerance - Successfully connected to another cluster
# member for state sharing
faulttolerance_stateok,

# Fault Tolerance - Reading state data from another cluster
# member failed
faulttolerance_statereadfail,

# Fault Tolerance - Timeout while sending state data to
# another cluster member
faulttolerance_statetimeout,

# Fault Tolerance - Received unexpected state data from
# another cluster member
faulttolerance_stateunexpected,

# Fault Tolerance - Writing state data to another cluster
# member failed
faulttolerance_statewritefail,

# Fault Tolerance - An error occurred when using the zcluster
# Multi-Hosted IP kernel module
faulttolerance_zclustermoderr,

# General - An error occurred during user authentication
general_autherror,
```

```
# General - Running out of free file descriptors
general_fewfreefds,

# General - Failed to load geolocation data.
general_geodataloadfail,

# General - Appliance hardware notification
general_hardware,

# General - Software must be restarted to apply configuration
# changes
general_restartrequired,

# General - Software is running
general_running,

# General - Time has been moved back
general_timemovedback,

# General - Zeus Traffic Manager software problem
general_zxtmswerror,

# Java - Java runner died
java_javadied,

# Java - Cannot start Java, program not found
java_javanotfound,

# Java - Java started
java_javastarted,

# Java - Java failed to start
java_javastartfail,

# Java - Java support has stopped
java_javastop,

# Java - Java failed to terminate
java_javaterminatefail,

# Java - Servlet encountered an error
java_servleterror,

# License Keys - License key bandwidth limit has been hit
licensekeys_bwlimited,

# License Keys - Configured cache size exceeds license limit,
# only using amount allowed by license
licensekeys_cachesizereduced,

# License Keys - License key has expired
```

```
licensekeys_expired,

# License Keys - License key expires within 4 days
licensekeys_expiresoon,

# License Keys - License allows less memory for caching
licensekeys_lessmemallowed,

# License Keys - License key authorized
licensekeys_license-authorized,

# License Keys - Unable to authorize license key
licensekeys_license-graceperiodexpired,

# License Keys - License server rejected license key; key
# remains authorized
licensekeys_license-rejected-authorized,

# License Keys - License server rejected license key; key is
# not authorized
licensekeys_license-rejected-unauthorized,

# License Keys - Unable to contact license server; license key
# remains authorized
licensekeys_license-timedout-authorized,

# License Keys - Unable to contact license server; license key
# is not authorized
licensekeys_license-timedout-unauthorized,

# License Keys - License key is not authorized
licensekeys_license-unauthorized,

# License Keys - Cluster size exceeds license key limit
licensekeys_licenseclustertoobig,

# License Keys - License key is corrupt
licensekeys_licensecorrupt,

# License Keys - License allows more memory for caching
licensekeys_morememallowed,

# License Keys - License key SSL transactions-per-second limit
# has been hit
licensekeys_ssltpslimited,

# License Keys - License key transactions-per-second limit has
# been hit
licensekeys_tpslimited,

# License Keys - Started without a license
```

```
licensekeys_unlicensed,

# License Keys - Using a development license
licensekeys_usingdevlicense,

# License Keys - Using license key
licensekeys_usinglicense,

# Monitors - Monitor has detected a failure
monitors_monitorfail,

# Monitors - Monitor is working
monitors_monitorok,

# Pools - HTTP response contained an invalid Content-Length
# header
pools_badcontentlen,

# Pools - Node returned invalid EHLO response
pools_ehloinvalid,

# Pools - Node has failed
pools_nodefail,

# Pools - Failed to resolve node address
pools_noderesolvefailure,

# Pools - Node resolves to multiple IP addresses
pools_noderesolvemultiple,

# Pools - Node is working again
pools_nodeworking,

# Pools - Node doesn't provide STARTTLS support
pools_nostarttls,

# Pools - Pool has no back-end nodes responding
pools_pooldied,

# Pools - Pool configuration contains no valid backend nodes
pools_poolnonodes,

# Pools - Pool now has working nodes
pools_poolok,

# Pools - Node returned invalid STARTTLS response
pools_starttlsinvalid,

# Service Protection Classes - Summary of recent service
# protection events
protection_triggersummary,
```

```
# Rules - data.set() has run out of space
rules_datastorefull,

# Rules - Rule selected an unresolvable host
rules_forwardproxybadhost,

# Rules - Rule used event.emit() with an invalid custom event
rules_invalidemit,

# Rules - Rule selected an unknown rate shaping class
rules_norate,

# Rules - Rule references an unknown pool via pool.activenodes
rules_poolactivenodesunknown,

# Rules - Rule selected an unknown pool
rules_pooluseunknown,

# Rules - Rule aborted during execution
rules_ruleabort,

# Rules - Rule encountered invalid data while uncompressing
# response
rules_rulebodycomperror,

# Rules - Rule has buffered more data than expected
rules_rulebufferlarge,

# Rules - Rule logged an info message using log.info
rules_rulelogmsginfo,

# Rules - Rule logged an error message using log.error
rules_rulelogmsgserious,

# Rules - Rule logged a warning message using log.warn
rules_rulelogmsgwarn,

# Rules - Rule selected an unknown session persistence class
rules_rulenopersistence,

# Rules - Client sent invalid HTTP request body
rules_rulesinvalidrequestbody,

# Rules - Attempt to use http.getResponse or
# http.getResponseBody after http.stream.startResponse
rules_rulestreamerrorgetresponse,

# Rules - Internal error while processing HTTP stream
rules_rulestreamerrorinternal,
```

```
# Rules - Rule did not supply enough data in HTTP stream
rules_rulestreamerrornotenough,

# Rules - Attempt to initialize HTTP stream before previous
# stream had finished
rules_rulestreamerrornotfinished,

# Rules - Attempt to stream data or finish a stream before
# streaming had been initialized
rules_rulestreamerrornotstarted,

# Rules - Data supplied to HTTP stream could not be processed
rules_rulestreamerrorprocessfailure,

# Rules - Rule supplied too much data in HTTP stream
rules_rulestreamerrortoomuch,

# Rules - Rule encountered an XML error
rules_rulexmlerr,

# SLM Classes - SLM shared class limit exceeded
slm_slmclasslimitexceeded,

# SLM Classes - SLM has fallen below serious threshold
slm_slmfallenbelowserious,

# SLM Classes - SLM has fallen below warning threshold
slm_slmfallenbelowwarn,

# SLM Classes - Node information when SLM is non-conforming
# (no SNMP trap)
slm_slmnodeinfo,

# SLM Classes - SLM has risen above the serious threshold
slm_slmrecoveredserious,

# SLM Classes - SLM has recovered
slm_slmrecoveredwarn,

# SSL Hardware - SSL hardware support failed
sslhw_sslhwfail,

# SSL Hardware - SSL hardware support restarted
sslhw_sslhwrestart,

# SSL Hardware - SSL hardware support started
sslhw_sslhwstart,

# Test event generated from the Zeus Administration Server.
testaction,
```

```
# Virtual Servers - A protocol error has occurred
vservers_connerror,

# Virtual Servers - A socket connection failure has occurred
vservers_connfail,

# Virtual Servers - A virtual server request log file was
# deleted (Zeus Appliances only)
vservers_logfiledeleted,

# Virtual Servers - Dropped connection, request exceeded
# max_client_buffer limit
vservers_maxclientbufferdrop,

# Virtual Servers - Pool uses a session persistence class that
# does not work with this virtual server's protocol
vservers_poolpersistencemismatch,

# Virtual Servers - Private key now OK
vservers_privkeyok,

# Virtual Servers - Error compressing HTTP response
vservers_respcompfail,

# Virtual Servers - Response from webserver too large
vservers_responsetoolarge,

# Virtual Servers - No suitable ports available for streaming
# data connection
vservers_rtspstreamnoports,

# Virtual Servers - No suitable ports available for streaming
# data connection
vservers_sipstreamnoports,

# Virtual Servers - Request(s) received while SSL
# configuration invalid, connection closed
vservers_ssldrop,

# Virtual Servers - One or more SSL connections from clients
# failed recently
vservers_sslfail,

# Virtual Servers - Certificate Authority certificate expired
vservers_vscacertexpired,

# Virtual Servers - Certificate Authority certificate will
# expire soon
vservers_vscacerttoexpire,

# Virtual Servers - CRL for a Certificate Authority is out of
```

```
    # date
    vservers_vscrloutofdate,

    # Virtual Servers - Failed to write log file for virtual
    # server
    vservers_vslogwritefail,

    # Virtual Servers - Public SSL certificate expired
    vservers_vssslcertexpired,

    # Virtual Servers - Public SSL certificate will expire soon
    vservers_vssslcerttoexpire,

    # Virtual Servers - Virtual server started
    vservers_vsstart,

    # Virtual Servers - Virtual server stopped
    vservers_vsstop,

    # Traffic Managers - Configuration update refused: traffic
    # manager version mismatch
    zxtms_versionmismatch
}
```

6.23 System.AccessLogs

URI: `http://soap.zeus.com/zxtm/1.0/System/AccessLogs/`

The AccessLogs interfaces provide operations on saved virtual server access logs for a Zeus Appliance. This interface is only available on a Zeus Appliance and is deprecated; use the System.RequestLogs interface instead.

6.23.1 Methods

deleteAllVSAccessLogs() throws InvalidOperation

Delete all the access logs for all virtual servers.

```
void deleteAllVSAccessLogs()
```

deleteVSAccessLog(logfiles) throws InvalidOperation, InvalidInput

Delete the specified access logs.

```
void deleteVSAccessLog(
   String[] logfiles
)
```

deleteVSAccessLogs(vservers) throws InvalidOperation

Delete the access logs for specific virtual servers.

```
void deleteVSAccessLogs(
   String[] vservers
)
```

getAllVSAccessLogs() throws InvalidOperation

Get the access logs for all virtual servers.

```
System.AccessLogs.VSAccessLog[] getAllVSAccessLogs()
```

getVSAccessLogs(vservers) throws InvalidOperation

Get the access logs for specific virtual servers.

```
System.AccessLogs.VSAccessLog[][] getVSAccessLogs(
   String[] vservers
)
```

6.23.2 Structures

System.AccessLogs.VSAccessLog

This structure contains the information for each virtual server access log.

```
struct System.AccessLogs.VSAccessLog {
   # The log filename.
   String filename;

   # The virtual server for this logfile.
   String virtual_server;

   # The date this logfile was created.
   Time logdate;

   # The size (in bytes) of this logfile.
   Integer filesize;
}
```

6.24 System.Cache

URI: `http://soap.zeus.com/zxtm/1.2/System/Cache/`

The System.Cache interface provides information about the content cache for a machine. Using this interface, you can retrieve both individual cache entries and global cache data,

delete all entries in the cache, delete entries matching wildcards or delete individual entries.

6.24.1 Methods

clearCacheContentItems(virtual_servers, protocols, hosts, items)

Delete individual items from the Web Cache. All input arguments are arrays of strings and only those items are deleted whose virtual server, protocol, host and path attribute match all the corresponding values for a given index into the arguments.

```
void clearCacheContentItems(
   String[] virtual_servers
   System.Cache.Protocol[] protocols
   String[] hosts
   String[] items
)
```

clearMatchingCacheContent(protocol, host_wildcard, path_wildcard)

Delete the Web Cache entries matching the input arguments.

```
void clearMatchingCacheContent(
   System.Cache.Protocol protocol
   String host_wildcard
   String path_wildcard
)
```

clearWebCache()

Clear all entries from the Web Cache for this machine.

```
void clearWebCache()
```

getCacheContent(protocol, host_wildcard, path_wildcard, max_entries)

Get information about the Web Cache entries matching the input arguments.

```
System.Cache.CacheContentInfo getCacheContent(
   System.Cache.Protocol protocol
   String host_wildcard
   String path_wildcard
   Integer max_entries
)
```

getGlobalCacheInfo()

Get the size of the Web Cache, the number of Web Cache entries and the percentage memory used by the Web Cache for this machine.

```
System.Cache.GlobalInfo getGlobalCacheInfo()
```

6.24.2 Structures

System.Cache.CacheContent

This structure contains the basic information about an individual cache entry for a machine.

```
struct System.Cache.CacheContent {
   # The virtual server hosting the entry.
   String virtual_server;

   # The protocol of the entry: http or https.
   System.Cache.Protocol protocol;

   # The host name of the entry.
   String host;

   # The path of the entry.
   String path;

   # The time that the entry was last used.
   Time time_used;

   # The time that the entry expires.
   Time time_expires;

   # The number of hits for the entry.
   Long hits;

   # The number of variants of this entry in the cache.
   Integer num_variants;

   # The HTTP response code for this entry in the cache.
   Integer response_code;

   # The HTTP versions the entry is cached for.
   String[] versions;

   # The set of request-header fields that determine if the cache
   # entry may be used for a particular request.
   String[] varys;
}
```

System.Cache.CacheContentInfo

This structure contains the information about the cache content.

```
struct System.Cache.CacheContentInfo {
```

```
   # The total number of items matching the wildcards in a query.
   Integer number_matching_items;

   # The total size of the items matching the wildcards in a
   # query.
   Long size_matching_items;

   # The set of individual entries in the cache that matched the
   # query.
   System.Cache.CacheContent[] matching_items;
}
```

System.Cache.GlobalInfo

This structure contains the basic information about the content cache for a machine.

```
struct System.Cache.GlobalInfo {
   # The number of bytes of memory used by the cache.
   Long bytes_used;

   # The percentage of the cache used.
   Float percent_used;

   # The number of entries in the cache.
   Integer entries;

   # The number of times a request has tried to get a page from
   # the cache.
   Long num_lookups;

   # The number of times a request has successfully been served
   # from the cache.
   Long num_hits;
}
```

6.24.3 Enumerations

System.Cache.Protocol

This enumeration defines the possible protocols for cache entries.

```
enum System.Cache.Protocol {
   # The hypertext transfer protocol (port 80 by default).
   http,

   # The hypertext transfer protocol secure (port 443 by
   # default).
   https,

   # This special value can be used as wildcard to match both
```

```
   # http and https. It is never returned by the methods in this
   # interface.
   both
}
```

6.25 System.Connections

URI: `http://soap.zeus.com/zxtm/1.0/System/Connections/`

The System.Connections interface provides information about the current and recent connections for this machine. Using this interface you can retrieve a list of all connections.

6.25.1 Methods

getAllConnections()

Get a list of all connections, current and recent, for this machine.

```
System.Connections.Connection[] getAllConnections()
```

6.25.2 Structures

System.Connections.Connection

This structure contains the basic information about a Connection. It is used when retrieving the current and recent connections for a machine.

```
struct System.Connections.Connection {
   # The source IP address and port for connection.
   String from;

   # The local IP address and port for connection.
   String via;

   # The destination node for the connection.
   String to;

   # The connection state.
   System.Connections.ConnectionState state;

   # The virtual server handling the request.
   String vserver;

   # The rule being executed.
   String rule;

   # The pool being used.
   String pool;
```

```
    # The number of bytes that were received from the client.
    Integer bytes_in;

    # The number of bytes that were sent to the client.
    Integer bytes_out;

    # The length of time that the connection has been established,
    # in seconds.
    Integer time_est;

    # The length of time since receiving the last client data, in
    # seconds.
    Integer time_client;

    # The length of time since receiving the last server data, in
    # seconds.
    Integer time_server;

    # The number of times that the connection to the node has been
    # retried.
    Integer retries;

    # The Service Level Monitoring class being used.
    String slm_class;

    # The Virtual Server Bandwidth class being used.
    String vs_bwclass;

    # The Pool Bandwidth class being used.
    String pool_bwclass;

    # The status code in the HTTP response.
    String code;

    # The host header/URL in the HTTP request.
    String request;
}
```

6.25.3 Enumerations

System.Connections.ConnectionState

This enumeration defines the possible states for a particular connection.

```
enum System.Connections.ConnectionState {
    # Current connection: reading data from the client ('R').
    reading_from_client,

    # Current connection: writing data to the client ('W').
    writing_to_client,
```

```
   # Current connection: executing rules against client request
   # ('X').
   executing_rule,

   # Current connection: connecting to a node ('c').
   connecting_to_node,

   # Current connection: writing data to a node ('w').
   writing_to_node,

   # Current connection: reading data from a node ('r').
   reading_from_node,

   # Current connection: closing connection with client ('C').
   closing_client_connection,

   # Current connection: holding connection with client in
   # keepalive state ('K').
   holding_client_connection,

   # Recent connection that is no longer active.
   recent_connection
}
```

6.26 System.LicenseKeys

URI: `http://soap.zeus.com/zxtm/1.0/System/LicenseKeys/`

The System.LicenseKeys interface provides license key information for this machine. Using this interface, you can add and delete license keys, and retrieve both the license key currently in use and a list of all existing license keys.

6.26.1 Methods

addLicenseKeys(license_texts) throws ObjectAlreadyExists, InvalidInput

Create and add each of the named license keys.

```
Integer[] addLicenseKeys(
   String[] license_texts
)
```

deleteLicenseKeys(serials) throws ObjectDoesNotExist

Delete each of the named license keys.

```
void deleteLicenseKeys(
   Integer[] serials
)
```

getAllLicenseKeys()

Get a list of all the serial numbers of the existing license keys.

```
Integer[] getAllLicenseKeys()
```

getCurrentLicenseKey()

Get the serial number of the license key currently being used by this machine.

```
Integer getCurrentLicenseKey()
```

getLicenseKeys(serials) throws ObjectDoesNotExist

For each of the named license keys, get the license key structure.

```
System.LicenseKeys.LicenseKey[] getLicenseKeys(
   Integer[] serials
)
```

getRawLicenseKeys(serials) throws ObjectDoesNotExist

For each of the named license keys, get the raw license key text.

```
String[] getRawLicenseKeys(
   Integer[] serials
)
```

6.26.2 Structures

System.LicenseKeys.LicenseKey

This structure contains the basic information for a license key. It is used when adding, deleting or retrieving license keys.

```
struct System.LicenseKeys.LicenseKey {
   # The name of the product the license is for.
   String product;

   # The traffic manager software version for this machine.
   String version;

   # The list of platforms that the software may run on.
   String[] platforms;

   # The maximum number of CPUs that the software may run on.
   # Note that this field may not exist for all license keys in
   # which case its value will be '0'.
   Integer maxcpus;
```

```
# The IP addresses of the machines that the software may run
# on. Note that this field may not exist for all license keys
# in which case its value will be the empty array.
String[] ip_address;

# The MAC addresses of the machines that the software may run
# on. Note that this field may not exist for all license keys
# in which case its value will be the empty array.
String[] mac_address;

# The features that are supported by the license key.
String[] features;

# The maximum number of backends supported by the license key.
# Note that this field may not exist for all license keys in
# which case its value will be '0'.
Integer max_backends;

# Additional customer information for the license key. Note
# that this field may not exist for all license keys in which
# case its value will be "".
String customer_info;

# The customer ID for the license key. Note that this field
# may not exist for all license keys in which case its value
# will be "".
String customer_id;

# The serial number of the license key.
Integer serial;

# The time at which the license key will expire.
Time expires;

# The time at which the license key was issued.
Time issued;

# The time at which the support contract for the license key
# expires. Note that this field is for future use so may not
# exist for all license keys, in which case its value will be
# equal to '01/01/1970 00:00:00'.
Time maintenance;

# The hardware serial number for the appliance with this
# license key. Note that this field is only applicable to Zeus
# Appliances and otherwise will have the value "".
String hwserial;

# The maximum cluster size supported by the license key. Note
# that this field may not exist for all license keys in which
# case its value will be equal to '0'.
```

```
   Integer cluster_size;
}
```

6.27 System.Log

URI: `http://soap.zeus.com/zxtm/1.0/System/Log/`

The System.Log interface provides audit log and error log information for this machine. Using this interface, you can retrieve the error log as a string, get a list of individual entries in the audit log and clear the error log.

6.27.1 Methods

clearErrorLog()

Clear the error log for this machine.

```
void clearErrorLog()
```

getAuditLog()

Get a list of the most recent elements of the audit log for this machine.

```
System.Log.AuditItem[] getAuditLog()
```

getAuditLogLines(max_lines)

Get a maximum of max_lines lines of the audit log for this machine.

```
System.Log.AuditItem[] getAuditLogLines(
   Integer max_lines
)
```

getErrorLogLines(max_lines)

Get a maximum of max_lines lines of the error log for this machine as a string, if max_lines is 0 then 1024 lines are returned.

```
String getErrorLogLines(
   Integer max_lines
)
```

getErrorLogString()

Get the error log for this machine as a string.

```
String getErrorLogString()
```

6.27.2 Structures

System.Log.AccessDenied

This is the operation parameters structure for 'accessdenied' operations (user denied access). It is used when getting Audit Log data.

```
struct System.Log.AccessDenied {
   # The name of the host that was denied access.
   String host;
}
```

System.Log.AddAuthenticator

This is the operation parameters structure for 'addauth' operations (authenticator added). It is used when getting Audit Log data.

```
struct System.Log.AddAuthenticator {
   # The name of the authenticator that was added.
   String modauth;

   # The type of the authenticator that was added.
   String authtype;
}
```

System.Log.AddFile

This is the operation parameters structure for 'addfile' operations (file added). It is used when getting Audit Log data.

```
struct System.Log.AddFile {
   # The name of the object that was added.
   String file;
}
```

System.Log.AddGroup

This is the operation parameters structure for 'addgroup' operations (group added). It is used when getting Audit Log data.

```
struct System.Log.AddGroup {
   # The name of the group that was added.
   String modgroup;
}
```

System.Log.AddUser

This is the operation parameters structure for 'adduser' operations (user added). It is used when getting Audit Log data.

```
struct System.Log.AddUser {
   # The name of the user that was added.
   String moduser;

   # The name of the group that the user was added to.
   String modgroup;
}
```

System.Log.Adhoc

This is the operation parameters structure for 'adhoc' operations (a custom event). It is used when getting Audit Log data.

```
struct System.Log.Adhoc {
   # The custom text used to describe the event.
   String text;

   # The (optional) name of the object used in the event.
   String obj;
}
```

System.Log.AuditItem

This structure contains the information about an event in the Audit Log file. It is used when getting Audit Log information.

```
struct System.Log.AuditItem {
   # The date and time at which the event occurred.
   Time date;

   # The name of the user who caused the event.
   String user;

   # The group of the user who caused the event.
   String group;

   # The authenticator that authorised the user who caused the
   # event.
   String auth;

   # The IP address of the user.
   String ip;

   # The type of operation that occurred.
   System.Log.OperationType op_type;

   # The list of parameters used in the operation. This list is
   # required for all operations with the exception of
   # LoginFail/Logout/Timeout operations for which there are no
   # additional parameters.
```

```
    System.Log.OpParam op_params;
}
```

System.Log.CopyAuthenticator

This is the operation parameters structure for 'copyauth' operations (authenticator copied). It is used when getting Audit Log data.

```
struct System.Log.CopyAuthenticator {
    # The name of the authenticator that was copied.
    String oldauth;

    # The name of the authenticator that was created.
    String modauth;

    # The authenticator type.
    String authtype;
}
```

System.Log.CopyFile

This is the operation parameters structure for 'copyfile' operations (file copied). It is used when getting Audit Log data.

```
struct System.Log.CopyFile {
    # The name of the new object.
    String file;

    # The name of the object on which the new object is based.
    String oldfile;
}
```

System.Log.CopyGroup

This is the operation parameters structure for 'copygroup' operations (group copied). It is used when getting Audit Log data.

```
struct System.Log.CopyGroup {
    # The name of the new group.
    String modgroup;

    # The name of the group on which the new group is based.
    String oldgroup;
}
```

System.Log.DeleteAuthenticator

This is the operation parameters structure for 'delauth' operations (authenticator deleted). It is used when getting Audit Log data.

```
struct System.Log.DeleteAuthenticator {
   # The name of the authenticator that was deleted.
   String modauth;

   # The type of the authenticator that was deleted.
   String authtype;
}
```

System.Log.DeleteFile

This is the operation parameters structure for 'delfile' operations (file deleted). It is used when getting Audit Log data.

```
struct System.Log.DeleteFile {
   # The name of the object that was deleted.
   String file;
}
```

System.Log.DeleteGroup

This is the operation parameters structure for 'delgroup' operations (group deleted). It is used when getting Audit Log data.

```
struct System.Log.DeleteGroup {
   # The name of the group that was deleted.
   String modgroup;
}
```

System.Log.DeleteUser

This is the operation parameters structure for 'deluser' operations (user deleted). It is used when getting Audit Log data.

```
struct System.Log.DeleteUser {
   # The name of the user that was deleted.
   String moduser;
}
```

System.Log.Login

This is the operation parameters structure for 'login' operations (user logged in). It is used when getting Audit Log data.

```
struct System.Log.Login {
   # The type of login.
   String logintype;

   # The user's timeout effective for the login session.
   String timeout;
```

```
}
```

System.Log.LoginFail

This is the operation parameters structure for 'loginfail' operations (login attempt failed). It is used when getting Audit Log data.

```
struct System.Log.LoginFail {
   # The type of login attempted.
   String logintype;

   # The resource the login was attempting to access.
   String resource;
}
```

System.Log.ModifyFile

This is the operation parameters structure for 'filemod' operations (a file other than a config key was modified). It is used when getting Audit Log data.

```
struct System.Log.ModifyFile {
   # The name of the object that was modified.
   String file;
}
```

System.Log.ModifyKey

This is the operation parameters structure for 'keymod' operations (config key modified). It is used when getting Audit Log data.

```
struct System.Log.ModifyKey {
   # The name of the object in which the modified key exists.
   String file;

   # The name of the key that was modified.
   String key;

   # The old value of the key.
   String oldval;

   # The new value of the key.
   String value;
}
```

System.Log.ModifyRule

This is the operation parameters structure for 'rulemod' operations (rule modified). It is used when getting Audit Log data.

```
struct System.Log.ModifyRule {
   # The name of the rule that was modified.
   String file;
}
```

System.Log.ModifyUser

This is the operation parameters structure for 'usermod' operations (user modified). It is used when getting Audit Log data.

```
struct System.Log.ModifyUser {
   # The name of the user that was added.
   String moduser;
}
```

System.Log.NoAccessPermission

This is the operation parameters structure for 'noperm' operations (no access permission). It is used when getting Audit Log data.

```
struct System.Log.NoAccessPermission {
   # The name of the UI section to which access was denied.
   String section;
}
```

System.Log.NoChangePermission

This is the operation parameters structure for 'nopostperm' operations (no permission to change data). It is used when getting Audit Log data.

```
struct System.Log.NoChangePermission {
   # The name of the UI section to which permission was denied.
   String section;
}
```

System.Log.OpParam

This is the base type structure for operation parameters. It is used when getting Audit Log data.

```
struct System.Log.OpParam {

}
```

System.Log.RenameFile

This is the operation parameters structure for 'renfile' operations (file renamed). It is used when getting Audit Log data.

```
struct System.Log.RenameFile implements System.Log.OpParam {
   # The new name of the object.
   String file;

   # The old name of the object.
   String oldfile;
}
```

System.Log.StartVS

This is the operation parameters structure for 'startvs' operations (start a Virtual Server). It is used when getting Audit Log data.

```
struct System.Log.StartVS {
   # The name of the Virtual Server that was started.
   String vs;
}
```

System.Log.StopVS

This is the operation parameters structure for 'stopvs' operations (stop a Virtual Server). It is used when getting Audit Log data.

```
struct System.Log.StopVS {
   # The name of the Virtual Server that was stopped.
   String vs;
}
```

6.27.3 Enumerations

System.Log.OperationType

This enumeration defines the possible types of operations that may exist in the audit log.

```
enum System.Log.OperationType {
   # An AddFile operation occurs when a file is added. This
   # operation is caused by a user creating a new object such as
   # a Virtual Server, Pool, etc. It appears as an 'addfile'
   # operation in the Audit Log.
   AddFile,

   # A DeleteFile operation occurs when a file is deleted. This
   # operation is caused by a user deleting an existing object
   # such as a Virtual Server, Pool, etc. It appears as a
   # 'delfile' operation in the Audit Log.
   DeleteFile,

   # A RenameFile operation occurs when a file is renamed. This
   # operation is caused by a user renaming an existing object
```

```
# such as a Virtual Server, Pool, etc. It appears as a
# 'renfile' operation in the Audit Log.
RenameFile,

# An AddGroup operation occurs when a new group of users is
# created. It appears as an 'addgroup' operation in the Audit
# Log.
AddGroup,

# A DeleteGroup operation occurs when a group of users is
# deleted. It appears as an 'delgroup' operation in the Audit
# Log.
DeleteGroup,

# A CopyGroup operation occurs when a user group is saved with
# a new group name. It appears as an 'copygroup' operation in
# the Audit Log.
CopyGroup,

# An AddUser operation occurs when a new user is added. It
# appears as an 'adduser' operation in the Audit Log.
AddUser,

# A DeleteUser operation occurs when an existing user is
# deleted. It appears as an 'deluser' operation in the Audit
# Log.
DeleteUser,

# A ModifyUser operation occurs when an existing user is
# modified. It appears as an 'usermod' operation in the Audit
# Log.
ModifyUser,

# A ModifyKey operation occurs when the value of a config file
# is modified. This operation is caused by a user changing the
# settings for an existing object such as a Virtual Server,
# Pool, etc. It appears as a 'keymod' operation in the Audit
# Log.
ModifyKey,

# A ModifyRule operation occurs when the contents of a rule
# are modified. This operation is caused by a user editing an
# existing rule. It appears as a 'rulemod' operation in the
# Audit Log.
ModifyRule,

# A ModifyFile operation occurs when the contents of a
# non-config file are modified. This differs from a ModifyKey
# operation in that ModifyFile operations are caused by the
# modification of non-config files which are not managed by
# the traffic manager, for example changing the settings of an
```

```
# SSL Certificate. It appears as a 'filemod' operation in the
# Audit Log.
ModifyFile,

# A CopyFile operation occurs when a file is copied. This
# operation is caused by the user saving an object as a new
# name, for example a Rule or an SSL Certificate. It appears
# as a 'copyfile' operation in the Audit Log.
CopyFile,

# An Adhoc operation represents a custom event which does not
# fit any of the other Operation Types. For example, it occurs
# when a user is adding or deleting a License Key or modifying
# the Security settings. It appears as an 'adhoc' operation
# the Audit Log.
Adhoc,

# An AccessDenied operation occurs when a user is denied
# access to the Admin Server due to access restrictions which
# are in place. It appears as an 'accessdenied' operation in
# the Audit Log.
AccessDenied,

# A NoAccessPermission operation occurs when a user is refused
# permission whist accessing a section of the Admin Server. It
# appears as a 'noperm' operation in the Audit Log.
NoAccessPermission,

# A NoChangePermission operation occurs when a user is refused
# permission to update data in a section of the Admin Server.
# It appears as a 'nopostperm' operation in the Audit Log.
NoChangePermission,

# A Login operation occurs when a user successfully logs on to
# the admin server. It appears as a 'login' operation in the
# Audit Log.
Login,

# A LoginFail operation occurs when a user tries and fails to
# log on to the admin server. This type of operation does not
# have any additional parameters to log therefore the
# 'op_params' field does not exist. It appears as a
# 'loginfail' operation in the Audit Log.
LoginFail,

# A Logout operation occurs when a user successfully logs out
# of the admin server. This type of operation does not have
# any additional parameters to log therefore the 'op_params'
# field does not exist. It appears as a 'logout' operation in
# the Audit Log.
Logout,
```

```
    # A Timeout operation occurs when a user session times out.
    # This type of operation does not have any additional
    # parameters to log therefore the 'op_params' field does not
    # exist. It appears as a 'timeout' operation in the Audit Log.
    Timeout,

    # A StartVS operation type occurs when a user starts an
    # existing virtual server. It appears as a 'startvs' operation
    # in the Audit Log.
    StartVS,

    # A StopVS operation type occurs when a user stops an existing
    # virtual server. It appears as a 'stopvs' operation in the
    # Audit Log.
    StopVS,

    # An AddAuthenticator operation type occurs when a new
    # authenticator is created. It appears as a 'addauth'
    # operation in the Audit Log.
    AddAuthenticator,

    # A DeleteAuthenticator operation type occurs when an existing
    # authenticator is deleted. It appears as a 'delauth'
    # operation in the Audit Log.
    DeleteAuthenticator,

    # A CopyAuthenticator operation type occurs when a new
    # authenticator is created by saving an existing authenticator
    # to a new name. It appears as a 'copyauth' operation in the
    # Audit Log.
    CopyAuthenticator
}
```

6.28 System.MachineInfo

URI: `http://soap.zeus.com/zxtm/1.0/System/MachineInfo/`

The System.MachineInfo interface provides information about the IP addresses, MAC addresses and traffic manager software version for this machine.

6.28.1 Methods

getAllClusterMachines()

Gets all of the machines in this traffic manager's cluster.

```
System.MachineInfo.Machine[] getAllClusterMachines()
```

getIPAddresses()

Get a list of IP addresses for this machine.

```
String[] getIPAddresses()
```

getMACAddresses()

Get a list of MAC addresses for this machine.

```
String[] getMACAddresses()
```

getProductVersion()

Get the traffic manager software version for this machine.

```
String getProductVersion()
```

getTrafficManagerUptime()

Get the time (in seconds) that the traffic manager has been running for.

```
Unsigned Integer getTrafficManagerUptime()
```

getZeusHome()

Get the install location of the traffic manager software (ZEUSHOME).

```
String getZeusHome()
```

isIPv6Enabled()

Check whether IPv6 is enabled on this system and supported by the traffic manager

```
Boolean isIPv6Enabled()
```

6.28.2 Structures

System.MachineInfo.Machine

This structure contains information about a traffic manager in the cluster.

```
struct System.MachineInfo.Machine {
   # The hostname of this machine
   String hostname;

   # The IP address of this machine.
   String ipaddress;
```

```
    # The URL of the admin server for this traffic manager.
    String admin_server;

    # The install path of the traffic manager on this machine.
    String zeushome;
}
```

6.29 System.RequestLogs

URI: `http://soap.zeus.com/zxtm/1.0/System/RequestLogs/`

The RequestLogs interfaces provide operations on saved virtual server request logs for a Zeus Appliance. This interface is only available on a Zeus Appliance.

6.29.1 Methods

deleteAllVSRequestLogs() throws InvalidOperation

Delete all the request logs for all virtual servers.

```
void deleteAllVSRequestLogs()
```

deleteVSRequestLog(logfiles) throws InvalidInput, InvalidOperation

Delete the specified request logs.

```
void deleteVSRequestLog(
   String[] logfiles
)
```

deleteVSRequestLogs(vservers) throws InvalidOperation

Delete the request logs for specific virtual servers.

```
void deleteVSRequestLogs(
   String[] vservers
)
```

getAllVSRequestLogs() throws InvalidOperation

Get the request logs for all virtual servers.

```
System.RequestLogs.VSRequestLog[] getAllVSRequestLogs()
```

getVSRequestLogs(vservers) throws InvalidOperation

Get the request logs for specific virtual servers.

```
System.RequestLogs.VSRequestLog[][] getVSRequestLogs(
   String[] vservers
)
```

6.29.2 Structures

System.RequestLogs.VSRequestLog

This structure contains the information for each virtual server request log.

```
struct System.RequestLogs.VSRequestLog {
   # The log filename.
   String filename;

   # The virtual server for this logfile.
   String virtual_server;

   # The date this logfile was created.
   Time logdate;

   # The size (in bytes) of this logfile.
   Integer filesize;
}
```

6.30 System.Stats

URI: `http://soap.zeus.com/zxtm/1.0/System/Stats/`

The System.Stats interface retrieves statistical information about the system. This interface is essentially an implementation of part of the SNMP interface in SOAP.

6.30.1 Methods

getActionNumber()

The number of actions configured in the traffic manager.

```
Integer getActionNumber()
```

getActions()

Gets the list of Alerting Actions configured.

```
String[] getActions()
```

getActionsProcessed(names) throws InvalidInput, InvalidObjectName

Number of times this action has been processed, for each of the named Actions.

```
Integer[] getActionsProcessed(
   String[] names
)
```

getAspSessionCacheEntries()

The total number of ASP sessions stored in the cache.

```
Integer getAspSessionCacheEntries()
```

getAspSessionCacheEntriesMax()

The maximum number of ASP sessions in the cache.

```
Integer getAspSessionCacheEntriesMax()
```

getAspSessionCacheHitRate()

The percentage of ASP session lookups that succeeded.

```
Integer getAspSessionCacheHitRate()
```

getAspSessionCacheHits()

Number of times a ASP session entry has been successfully found in the cache.

```
Integer getAspSessionCacheHits()
```

getAspSessionCacheLookups()

Number of times a ASP session entry has been looked up in the cache.

```
Integer getAspSessionCacheLookups()
```

getAspSessionCacheMisses()

Number of times a ASP session entry has not been available in the cache.

```
Integer getAspSessionCacheMisses()
```

getAspSessionCacheOldest()

The age of the oldest ASP session in the cache (in seconds).

```
Integer getAspSessionCacheOldest()
```

getBandwidthClassBytesOut(names) throws InvalidInput, InvalidObjectName

Bytes output by connections assigned to this bandwidth class, for each of the named BandwidthClasses.

```
Long[] getBandwidthClassBytesOut(
   String[] names
)
```

getBandwidthClassGuarantee(names) throws InvalidInput, InvalidObjectName

Guaranteed bandwidth class limit (kbits/s). Currently unused, for each of the named BandwidthClasses.

```
Integer[] getBandwidthClassGuarantee(
   String[] names
)
```

getBandwidthClassMaximum(names) throws InvalidInput, InvalidObjectName

Maximum bandwidth class limit (kbits/s), for each of the named BandwidthClasses.

```
Integer[] getBandwidthClassMaximum(
   String[] names
)
```

getBandwidthClassNumber()

The number of bandwidth classes defined.

```
Integer getBandwidthClassNumber()
```

getBandwidthClasses()

Gets the list of Bandwidth Classes configured.

```
String[] getBandwidthClasses()
```

getDataEntries()

Number of entries in the TrafficScript data.get()/set() storage.

```
Integer getDataEntries()
```

getDataMemoryUsage()

Number of bytes used in the TrafficScript data.get()/set() storage.

```
Integer getDataMemoryUsage()
```

getEventNumber()

The number of event configurations.

```
Integer getEventNumber()
```

getEvents()

Gets the list of Event Types configured.

```
String[] getEvents()
```

getEventsMatched(names) throws InvalidInput, InvalidObjectName

Number of times this event configuration has matched, for each of the named Events.

```
Integer[] getEventsMatched(
   String[] names
)
```

getEventsSeen()

Events seen by the traffic Manager's event handling process.

```
Integer getEventsSeen()
```

getInterfaceCollisions(names) throws InvalidInput, InvalidObjectName

The number of collisions reported by this interface, for each of the named Interfaces.

```
Integer[] getInterfaceCollisions(
   String[] names
)
```

getInterfaceNumber()

The number of network interfaces.

```
Integer getInterfaceNumber()
```

getInterfaceRxBytes(names) throws InvalidInput, InvalidObjectName

Bytes received by this interface, for each of the named Interfaces.

```
Long[] getInterfaceRxBytes(
   String[] names
)
```

getInterfaceRxErrors(names) throws InvalidInput, InvalidObjectName

The number of receive errors reported by this interface, for each of the named Interfaces.

```
Integer[] getInterfaceRxErrors(
   String[] names
)
```

getInterfaceRxPackets(names) throws InvalidInput, InvalidObjectName

The number of packets received by this interface, for each of the named Interfaces.

```
Integer[] getInterfaceRxPackets(
   String[] names
)
```

getInterfaceTxBytes(names) throws InvalidInput, InvalidObjectName

Bytes transmitted by this interface, for each of the named Interfaces.

```
Long[] getInterfaceTxBytes(
   String[] names
)
```

getInterfaceTxErrors(names) throws InvalidInput, InvalidObjectName

The number of transmit errors reported by this interface, for each of the named Interfaces.

```
Integer[] getInterfaceTxErrors(
   String[] names
)
```

getInterfaceTxPackets(names) throws InvalidInput, InvalidObjectName

The number of packets transmitted by this interface, for each of the named Interfaces.

```
Integer[] getInterfaceTxPackets(
   String[] names
)
```

getInterfaces()

Gets the list of Network Interfaces configured.

```
String[] getInterfaces()
```

getIpSessionCacheEntries()

The total number of IP sessions stored in the cache.

```
Integer getIpSessionCacheEntries()
```

getIpSessionCacheEntriesMax()

The maximum number of IP sessions in the cache.

```
Integer getIpSessionCacheEntriesMax()
```

getIpSessionCacheHitRate()

The percentage of IP session lookups that succeeded.

```
Integer getIpSessionCacheHitRate()
```

getIpSessionCacheHits()

Number of times a IP session entry has been successfully found in the cache.

```
Integer getIpSessionCacheHits()
```

getIpSessionCacheLookups()

Number of times a IP session entry has been looked up in the cache.

```
Integer getIpSessionCacheLookups()
```

getIpSessionCacheMisses()

Number of times a IP session entry has not been available in the cache.

```
Integer getIpSessionCacheMisses()
```

getIpSessionCacheOldest()

The age of the oldest IP session in the cache (in seconds).

```
Integer getIpSessionCacheOldest()
```

getJ2eeSessionCacheEntries()

The total number of J2EE sessions stored in the cache.

```
Integer getJ2eeSessionCacheEntries()
```

getJ2eeSessionCacheEntriesMax()

The maximum number of J2EE sessions in the cache.

```
Integer getJ2eeSessionCacheEntriesMax()
```

getJ2eeSessionCacheHitRate()

The percentage of J2EE session lookups that succeeded.

```
Integer getJ2eeSessionCacheHitRate()
```

getJ2eeSessionCacheHits()

Number of times a J2EE session entry has been successfully found in the cache.

```
Integer getJ2eeSessionCacheHits()
```

getJ2eeSessionCacheLookups()

Number of times a J2EE session entry has been looked up in the cache.

```
Integer getJ2eeSessionCacheLookups()
```

getJ2eeSessionCacheMisses()

Number of times a J2EE session entry has not been available in the cache.

```
Integer getJ2eeSessionCacheMisses()
```

getJ2eeSessionCacheOldest()

The age of the oldest J2EE session in the cache (in seconds).

```
Integer getJ2eeSessionCacheOldest()
```

getLicensekeyNumber()

The number of License keys.

```
Integer getLicensekeyNumber()
```

getMonitorNumber()

The number of Monitors.

```
Integer getMonitorNumber()
```

getNodeBytesFromNode(nodes) throws InvalidInput, InvalidObjectName

Bytes received from this node, for each of the specified Nodes.

```
Long[] getNodeBytesFromNode(
   System.Stats.Node[] nodes
)
```

getNodeBytesToNode(nodes) throws InvalidInput, InvalidObjectName

Bytes sent to this node, for each of the specified Nodes.

```
Long[] getNodeBytesToNode(
   System.Stats.Node[] nodes
)
```

getNodeCurrentConn(nodes) throws InvalidInput, InvalidObjectName

Connections currently established to this node, for each of the specified Nodes.

```
Integer[] getNodeCurrentConn(
   System.Stats.Node[] nodes
)
```

getNodeErrors(nodes) throws InvalidInput, InvalidObjectName

Number of timeouts, connection problems and other errors for this node, for each of the specified Nodes.

```
Integer[] getNodeErrors(
   System.Stats.Node[] nodes
)
```

getNodeFailures(nodes) throws InvalidInput, InvalidObjectName

Failures of this node, for each of the specified Nodes.

```
Integer[] getNodeFailures(
   System.Stats.Node[] nodes
)
```

getNodeIdleConns(nodes) throws InvalidInput, InvalidObjectName

Number of idle HTTP connections to this node, for each of the specified Nodes.

```
Integer[] getNodeIdleConns(
   System.Stats.Node[] nodes
)
```

getNodeNewConn(nodes) throws InvalidInput, InvalidObjectName

Requests that created a new connection to this node, for each of the specified Nodes.

```
Integer[] getNodeNewConn(
   System.Stats.Node[] nodes
)
```

getNodeNumber()

The number of nodes on this system (includes IPv4 and IPv6 nodes).

```
Integer getNodeNumber()
```

getNodePooledConn(nodes) throws InvalidInput, InvalidObjectName

Requests that reused an existing pooled/keepalive connection rather than creating a new TCP connection, for each of the specified Nodes.

```
Integer[] getNodePooledConn(
   System.Stats.Node[] nodes
)
```

getNodeResponseMax(nodes) throws InvalidInput, InvalidObjectName

Maximum response time (ms) in the last second for this node, for each of the specified Nodes.

```
Integer[] getNodeResponseMax(
   System.Stats.Node[] nodes
)
```

getNodeResponseMean(nodes) throws InvalidInput, InvalidObjectName

Mean response time (ms) in the last second for this node, for each of the specified Nodes.

```
Integer[] getNodeResponseMean(
   System.Stats.Node[] nodes
)
```

getNodeResponseMin(nodes) throws InvalidInput, InvalidObjectName

Minimum response time (ms) in the last second for this node, for each of the specified Nodes.

```
Integer[] getNodeResponseMin(
   System.Stats.Node[] nodes
)
```

getNodeState(nodes) throws InvalidInput, InvalidObjectName

The state of this node, for each of the specified Nodes.

```
System.Stats.NodeState[] getNodeState(
   System.Stats.Node[] nodes
)
```

getNodeTotalConn(nodes) throws InvalidInput, InvalidObjectName

Requests sent to this node, for each of the specified Nodes.

```
Integer[] getNodeTotalConn(
   System.Stats.Node[] nodes
)
```

getNodes()

Retrieves the list of available Nodes.

```
System.Stats.Node[] getNodes()
```

getNumIdleConnections()

Total number of idle HTTP connections to all nodes (used for future HTTP requests).

```
Integer getNumIdleConnections()
```

getNumberChildProcesses()

The number of traffic manager child processes.

```
Integer getNumberChildProcesses()
```

getNumberDNSACacheHits()

Requests for DNS A records resolved from the traffic manager's local cache.

```
Integer getNumberDNSACacheHits()
```

getNumberDNSARequests()

Requests for DNS A records (hostname->IP address) made by the traffic manager.

```
Integer getNumberDNSARequests()
```

getNumberDNSPTRCacheHits()

Requests for DNS PTR records resolved from the traffic manager's local cache.

```
Integer getNumberDNSPTRCacheHits()
```

getNumberDNSPTRRequests()

Requests for DNS PTR records (IP address->hostname) made by the traffic manager.

```
Integer getNumberDNSPTRRequests()
```

getNumberSNMPBadRequests()

Malformed SNMP requests received.

```
Integer getNumberSNMPBadRequests()
```

getNumberSNMPGetNextRequests()

SNMP GetNextRequests received.

```
Integer getNumberSNMPGetNextRequests()
```

getNumberSNMPGetRequests()

SNMP GetRequests received.

```
Integer getNumberSNMPGetRequests()
```

getNumberSNMPUnauthorisedRequests()

SNMP requests dropped due to access restrictions.

```
Integer getNumberSNMPUnauthorisedRequests()
```

getPerNodeServiceLevelResponseMax(per_node_service_levels) throws InvalidInput, InvalidObjectName

Maximum response time (ms) in the last second for this SLM class to this node, for each of the specified PerNodeServiceLevels.

```
Integer[] getPerNodeServiceLevelResponseMax(
   System.Stats.PerNodeServiceLevel[] per_node_service_levels
)
```

getPerNodeServiceLevelResponseMean(per_node_service_levels) throws InvalidInput, InvalidObjectName

Mean response time (ms) in the last second for this SLM class to this node, for each of the specified PerNodeServiceLevels.

```
Integer[] getPerNodeServiceLevelResponseMean(
   System.Stats.PerNodeServiceLevel[] per_node_service_levels
)
```

getPerNodeServiceLevelResponseMin(per_node_service_levels) throws InvalidInput, InvalidObjectName

Minimum response time (ms) in the last second for this SLM class to this node, for each of the specified PerNodeServiceLevels.

```
Integer[] getPerNodeServiceLevelResponseMin(
   System.Stats.PerNodeServiceLevel[] per_node_service_levels
)
```

getPerNodeServiceLevelTotalConn(per_node_service_levels) throws InvalidInput, InvalidObjectName

Requests handled by this SLM class to this node, for each of the specified PerNodeServiceLevels.

```
Integer[] getPerNodeServiceLevelTotalConn(
   System.Stats.PerNodeServiceLevel[] per_node_service_levels
)
```

getPerNodeServiceLevelTotalNonConf(per_node_service_levels) throws InvalidInput, InvalidObjectName

Non-conforming requests handled by this SLM class to this node, for each of the specified PerNodeServiceLevels.

```
Integer[] getPerNodeServiceLevelTotalNonConf(
   System.Stats.PerNodeServiceLevel[] per_node_service_levels
)
```

getPerNodeServiceLevels()

Retrieves the list of available PerNodeServiceLevels.

```
System.Stats.PerNodeServiceLevel[] getPerNodeServiceLevels()
```

getPerPoolNodeBytesFromNode(per_pool_nodes) throws InvalidInput, InvalidObjectName

Bytes received from this node, for each of the specified PerPoolNodes.

```
Long[] getPerPoolNodeBytesFromNode(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeBytesToNode(per_pool_nodes) throws InvalidInput, InvalidObjectName

Bytes sent to this node, for each of the specified PerPoolNodes.

```
Long[] getPerPoolNodeBytesToNode(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeCurrentConn(per_pool_nodes) throws InvalidInput, InvalidObjectName

Connections currently established to this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeCurrentConn(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeErrors(per_pool_nodes) throws InvalidInput, InvalidObjectName

Number of timeouts, connection problems and other errors for this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeErrors(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeFailures(per_pool_nodes) throws InvalidInput, InvalidObjectName

Failures of this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeFailures(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeIdleConns(per_pool_nodes) throws InvalidInput, InvalidObjectName

Number of idle HTTP connections to this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeIdleConns(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeNewConn(per_pool_nodes) throws InvalidInput, InvalidObjectName

Requests that created a new connection to this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeNewConn(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeNumber()

The number of nodes on this system.

```
Integer getPerPoolNodeNumber()
```

getPerPoolNodePooledConn(per_pool_nodes) throws InvalidInput, InvalidObjectName

Requests that reused an existing pooled/keepalive connection rather than creating a new TCP connection, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodePooledConn(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeResponseMax(per_pool_nodes) throws InvalidInput, InvalidObjectName

Maximum response time (ms) in the last second for this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeResponseMax(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeResponseMean(per_pool_nodes) throws InvalidInput, InvalidObjectName

Mean response time (ms) in the last second for this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeResponseMean(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeResponseMin(per_pool_nodes) throws InvalidInput, InvalidObjectName

Minimum response time (ms) in the last second for this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeResponseMin(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeState(per_pool_nodes) throws InvalidInput, InvalidObjectName

The state of this node, for each of the specified PerPoolNodes.

```
System.Stats.PerPoolNodeState[] getPerPoolNodeState(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodeTotalConn(per_pool_nodes) throws InvalidInput, InvalidObjectName

Requests sent to this node, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeTotalConn(
   System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodes()

Retrieves the list of available PerPoolNodes.

```
System.Stats.PerPoolNode[] getPerPoolNodes()
```

getPoolAlgorithm(names) throws InvalidInput, InvalidObjectName

The load-balancing algorithm the pool uses, for each of the named Pools.

```
System.Stats.PoolAlgorithm[] getPoolAlgorithm(
   String[] names
)
```

getPoolBytesIn(names) throws InvalidInput, InvalidObjectName

Bytes received by this pool from nodes, for each of the named Pools.

```
Long[] getPoolBytesIn(
   String[] names
)
```

getPoolBytesOut(names) throws InvalidInput, InvalidObjectName

Bytes sent by this pool to nodes, for each of the named Pools.

```
Long[] getPoolBytesOut(
   String[] names
)
```

getPoolDisabled(names) throws InvalidInput, InvalidObjectName

The number of nodes in this pool that are disabled, for each of the named Pools.

```
Integer[] getPoolDisabled(
   String[] names
)
```

getPoolDraining(names) throws InvalidInput, InvalidObjectName

The number of nodes in this pool which are draining, for each of the named Pools.

```
Integer[] getPoolDraining(
   String[] names
)
```

getPoolNodes(names) throws InvalidInput, InvalidObjectName

The number of nodes registered with this pool, for each of the named Pools.

```
Integer[] getPoolNodes(
   String[] names
)
```

getPoolNumber()

The number of pools on this system.

```
Integer getPoolNumber()
```

getPoolPersistence(names) throws InvalidInput, InvalidObjectName

The session persistence method this pool uses, for each of the named Pools.

```
System.Stats.PoolPersistence[] getPoolPersistence(
   String[] names
)
```

getPoolSessionMigrated(names) throws InvalidInput, InvalidObjectName

Sessions migrated to a new node because the desired node was unavailable, for each of the named Pools.

```
Integer[] getPoolSessionMigrated(
   String[] names
)
```

getPoolState(names) throws InvalidInput, InvalidObjectName

The state of this pool, for each of the named Pools.

```
System.Stats.PoolState[] getPoolState(
   String[] names
)
```

getPoolTotalConn(names) throws InvalidInput, InvalidObjectName

Requests sent to this pool, for each of the named Pools.

```
Integer[] getPoolTotalConn(
   String[] names
)
```

getPools()

Gets the list of Pools configured.

```
String[] getPools()
```

getRateClassConnsEntered(names) throws InvalidInput, InvalidObjectName

Connections that have entered the rate class and have been queued, for each of the named RateClasses.

```
Integer[] getRateClassConnsEntered(
   String[] names
)
```

getRateClassConnsLeft(names) throws InvalidInput, InvalidObjectName

Connections that have left the rate class, for each of the named RateClasses.

```
Integer[] getRateClassConnsLeft(
   String[] names
)
```

getRateClassCurrentRate(names) throws InvalidInput, InvalidObjectName

The average rate that requests are passing through this rate class, for each of the named RateClasses.

```
Integer[] getRateClassCurrentRate(
   String[] names
)
```

getRateClassDropped(names) throws InvalidInput, InvalidObjectName

Requests dropped from this rate class without being processed (e.g. timeouts), for each of the named RateClasses.

```
Integer[] getRateClassDropped(
   String[] names
)
```

getRateClassMaxRatePerMin(names) throws InvalidInput, InvalidObjectName

The maximum rate that requests may pass through this rate class (requests/min), for each of the named RateClasses.

```
Integer[] getRateClassMaxRatePerMin(
   String[] names
)
```

getRateClassMaxRatePerSec(names) throws InvalidInput, InvalidObjectName

The maximum rate that requests may pass through this rate class (requests/sec), for each of the named RateClasses.

```
Integer[] getRateClassMaxRatePerSec(
   String[] names
)
```

getRateClassNumber()

The number of rate classes defined.

```
Integer getRateClassNumber()
```

getRateClassQueueLength(names) throws InvalidInput, InvalidObjectName

The current number of requests queued by this rate class, for each of the named RateClasses.

```
Integer[] getRateClassQueueLength(
   String[] names
)
```

getRateClasses()

Gets the list of Rate Classes configured.

```
String[] getRateClasses()
```

getRuleAborts(names) throws InvalidInput, InvalidObjectName

Number of times this TrafficScript rule has aborted, for each of the named Rules.

```
Integer[] getRuleAborts(
   String[] names
)
```

getRuleDiscards(names) throws InvalidInput, InvalidObjectName

Number of times this TrafficScript rule has discarded the connection, for each of the named Rules.

```
Integer[] getRuleDiscards(
   String[] names
)
```

getRuleExecutions(names) throws InvalidInput, InvalidObjectName

Number of times this TrafficScript rule has been executed, for each of the named Rules.

```
Integer[] getRuleExecutions(
   String[] names
)
```

getRuleNumber()

The number of TrafficScript rules.

```
Integer getRuleNumber()
```

getRulePoolSelect(names) throws InvalidInput, InvalidObjectName

Number of times this TrafficScript rule has selected a pool to use, for each of the named Rules.

```
Integer[] getRulePoolSelect(
   String[] names
)
```

getRuleResponds(names) throws InvalidInput, InvalidObjectName

Number of times this TrafficScript rule has responded directly to the client, for each of the named Rules.

```
Integer[] getRuleResponds(
   String[] names
)
```

getRuleRetries(names) throws InvalidInput, InvalidObjectName

Number of times this TrafficScript rule has forced the request to be retried, for each of the named Rules.

```
Integer[] getRuleRetries(
   String[] names
)
```

getRules()

Gets the list of Rules configured.

```
String[] getRules()
```

getServiceLevelConforming(names) throws InvalidInput, InvalidObjectName

Percentage of requests associated with this SLM class that are conforming, for each of the named ServiceLevels.

```
Integer[] getServiceLevelConforming(
   String[] names
)
```

getServiceLevelCurrentConns(names) throws InvalidInput, InvalidObjectName

The number of connections currently associated with this SLM class, for each of the named ServiceLevels.

```
Integer[] getServiceLevelCurrentConns(
   String[] names
```

```
)
```

getServiceLevelIsOK(names) throws InvalidInput, InvalidObjectName

Indicates if this SLM class is currently conforming, for each of the named ServiceLevels.

```
System.Stats.ServiceLevelIsOK[] getServiceLevelIsOK(
   String[] names
)
```

getServiceLevelNumber()

The number of SLM classes defined.

```
Integer getServiceLevelNumber()
```

getServiceLevelResponseMax(names) throws InvalidInput, InvalidObjectName

Maximum response time (ms) in the last second for this SLM class, for each of the named ServiceLevels.

```
Integer[] getServiceLevelResponseMax(
   String[] names
)
```

getServiceLevelResponseMean(names) throws InvalidInput, InvalidObjectName

Mean response time (ms) in the last second for this SLM class, for each of the named ServiceLevels.

```
Integer[] getServiceLevelResponseMean(
   String[] names
)
```

getServiceLevelResponseMin(names) throws InvalidInput, InvalidObjectName

Minimum response time (ms) in the last second for this SLM class, for each of the named ServiceLevels.

```
Integer[] getServiceLevelResponseMin(
   String[] names
)
```

getServiceLevelTotalConn(names) throws InvalidInput, InvalidObjectName

Requests handled by this SLM class, for each of the named ServiceLevels.

```
Integer[] getServiceLevelTotalConn(
   String[] names
)
```

getServiceLevelTotalNonConf(names) throws InvalidInput, InvalidObjectName

Non-conforming requests handled by this SLM class, for each of the named ServiceLevels.

```
Integer[] getServiceLevelTotalNonConf(
   String[] names
)
```

getServiceLevels()

Gets the list of Service Level Monitoring classes configured.

```
String[] getServiceLevels()
```

getServiceProtLastRefusalTime(names) throws InvalidInput, InvalidObjectName

The time (in hundredths of a second) since this service protection class last refused a connection (this value will wrap if no connections are refused in more than 497 days), for each of the named ServiceProts.

```
Integer[] getServiceProtLastRefusalTime(
   String[] names
)
```

getServiceProtNumber()

The number of service protection classes defined.

```
Integer getServiceProtNumber()
```

getServiceProtRefusalBinary(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the request contained disallowed binary content, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalBinary(
   String[] names
)
```

getServiceProtRefusalConc10IP(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the top 10 source IP addresses issued too many concurrent connections, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalConc10IP(
   String[] names
)
```

getServiceProtRefusalConc1IP(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the source IP address issued too many concurrent connections, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalConc1IP(
   String[] names
)
```

getServiceProtRefusalConnRate(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the source IP address issued too many connections within 60 seconds, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalConnRate(
   String[] names
)
```

getServiceProtRefusalIP(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the source IP address was banned, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalIP(
   String[] names
)
```

getServiceProtRefusalRFC2396(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the HTTP request was not RFC 2396 compliant, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalRFC2396(
   String[] names
)
```

getServiceProtRefusalSize(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class because the request was larger than the defined limits allowed, for each of the named ServiceProts.

```
Integer[] getServiceProtRefusalSize(
   String[] names
)
```

getServiceProtTotalRefusal(names) throws InvalidInput, InvalidObjectName

Connections refused by this service protection class, for each of the named ServiceProts.

```
Integer[] getServiceProtTotalRefusal(
   String[] names
)
```

getServiceProts()

Gets the list of Service Protection Classes configured.

```
String[] getServiceProts()
```

getSslCacheEntries()

The total number of SSL sessions stored in the server cache.

```
Integer getSslCacheEntries()
```

getSslCacheEntriesMax()

The maximum number of SSL entries in the server cache.

```
Integer getSslCacheEntriesMax()
```

getSslCacheHitRate()

The percentage of SSL server cache lookups that succeeded.

```
Integer getSslCacheHitRate()
```

getSslCacheHits()

Number of times a SSL entry has been successfully found in the server cache.

```
Integer getSslCacheHits()
```

getSslCacheLookups()

Number of times a SSL entry has been looked up in the server cache.

```
Integer getSslCacheLookups()
```

getSslCacheMisses()

Number of times a SSL entry has not been available in the server cache.

```
Integer getSslCacheMisses()
```

getSslCacheOldest()

The age of the oldest SSL session in the server cache (in seconds).

```
Integer getSslCacheOldest()
```

getSslCipher3DESDecrypts()

Bytes decrypted with 3DES.

```
Integer getSslCipher3DESDecrypts()
```

getSslCipher3DESEncrypts()

Bytes encrypted with 3DES.

```
Integer getSslCipher3DESEncrypts()
```

getSslCipherAESDecrypts()

Bytes decrypted with AES.

```
Integer getSslCipherAESDecrypts()
```

getSslCipherAESEncrypts()

Bytes encrypted with AES.

```
Integer getSslCipherAESEncrypts()
```

getSslCipherDESDecrypts()

Bytes decrypted with DES.

```
Integer getSslCipherDESDecrypts()
```

getSslCipherDESEncrypts()

Bytes encrypted with DES.

```
Integer getSslCipherDESEncrypts()
```

getSslCipherDecrypts()

Bytes decrypted with a symmetric cipher.

```
Integer getSslCipherDecrypts()
```

getSslCipherEncrypts()

Bytes encrypted with a symmetric cipher.

```
Integer getSslCipherEncrypts()
```

getSslCipherRC4Decrypts()

Bytes decrypted with RC4.

```
Integer getSslCipherRC4Decrypts()
```

getSslCipherRC4Encrypts()

Bytes encrypted with RC4.

```
Integer getSslCipherRC4Encrypts()
```

getSslCipherRSADecrypts()

Number of RSA decrypts.

```
Integer getSslCipherRSADecrypts()
```

getSslCipherRSADecryptsExternal()

Number of external RSA decrypts.

```
Integer getSslCipherRSADecryptsExternal()
```

getSslCipherRSAEncrypts()

Number of RSA encrypts.

```
Integer getSslCipherRSAEncrypts()
```

getSslCipherRSAEncryptsExternal()

Number of external RSA encrypts.

```
Integer getSslCipherRSAEncryptsExternal()
```

getSslClientCertExpired()

Number of times a client certificate has expired.

```
Integer getSslClientCertExpired()
```

getSslClientCertInvalid()

Number of times a client certificate was invalid.

```
Integer getSslClientCertInvalid()
```

getSslClientCertNotSent()

Number of times a client certificate was required but not supplied.

```
Integer getSslClientCertNotSent()
```

getSslClientCertRevoked()

Number of times a client certificate was revoked.

```
Integer getSslClientCertRevoked()
```

getSslConnections()

Number of SSL connections negotiated.

```
Integer getSslConnections()
```

getSslHandshakeSSLv2()

Number of SSLv2 handshakes.

```
Integer getSslHandshakeSSLv2()
```

getSslHandshakeSSLv3()

Number of SSLv3 handshakes.

```
Integer getSslHandshakeSSLv3()
```

getSslHandshakeTLSv1()

Number of TLSv1.0 handshakes.

```
Integer getSslHandshakeTLSv1()
```

getSslHandshakeTLSv11()

Number of TLSv1.1 handshakes.

```
Integer getSslHandshakeTLSv11()
```

getSslSessionCacheEntries()

The total number of SSL session persistence entries stored in the cache.

```
Integer getSslSessionCacheEntries()
```

getSslSessionCacheEntriesMax()

The maximum number of SSL session persistence entries in the cache.

```
Integer getSslSessionCacheEntriesMax()
```

getSslSessionCacheHitRate()

The percentage of SSL session persistence lookups that succeeded.

```
Integer getSslSessionCacheHitRate()
```

getSslSessionCacheHits()

Number of times a SSL session persistence entry has been successfully found in the cache.

```
Integer getSslSessionCacheHits()
```

getSslSessionCacheLookups()

Number of times a SSL session persistence entry has been looked up in the cache.

```
Integer getSslSessionCacheLookups()
```

getSslSessionCacheMisses()

Number of times a SSL session persistence entry has not been available in the cache.

```
Integer getSslSessionCacheMisses()
```

getSslSessionCacheOldest()

The age of the oldest SSL session in the cache (in seconds).

```
Integer getSslSessionCacheOldest()
```

getSslSessionIDDiskCacheHit()

Number of times the SSL session id was found in the disk cache and reused (deprecated, will always return 0).

```
Integer getSslSessionIDDiskCacheHit()
```

getSslSessionIDDiskCacheMiss()

Number of times the SSL session id was not found in the disk cache (deprecated, will always return 0).

```
Integer getSslSessionIDDiskCacheMiss()
```

getSslSessionIDMemCacheHit()

Number of times the SSL session id was found in the cache and reused.

```
Integer getSslSessionIDMemCacheHit()
```

getSslSessionIDMemCacheMiss()

Number of times the SSL session id was not found in the cache.

```
Integer getSslSessionIDMemCacheMiss()
```

getSysCPUBusyPercent()

Percentage of time that the CPUs are busy.

```
Integer getSysCPUBusyPercent()
```

getSysCPUIdlePercent()

Percentage of time that the CPUs are idle.

```
Integer getSysCPUIdlePercent()
```

getSysCPUSystemBusyPercent()

Percentage of time that the CPUs are busy running system code.

```
Integer getSysCPUSystemBusyPercent()
```

getSysCPUUserBusyPercent()

Percentage of time that the CPUs are busy running user-space code.

```
Integer getSysCPUUserBusyPercent()
```

getSysFDsFree()

Number of free file descriptors.

```
Integer getSysFDsFree()
```

getSysMemBuffered()

Buffer memory (MBytes).

```
Integer getSysMemBuffered()
```

getSysMemFree()

Free memory (MBytes).

```
Integer getSysMemFree()
```

getSysMemInUse()

Memory used (MBytes).

```
Integer getSysMemInUse()
```

getSysMemSwapTotal()

Total swap space (MBytes).

```
Integer getSysMemSwapTotal()
```

getSysMemSwapped()

Amount of swap space in use (MBytes).

```
Integer getSysMemSwapped()
```

getSysMemTotal()

Total memory (MBytes).

```
Integer getSysMemTotal()
```

getTimeLastConfigUpdate()

The time (in hundredths of a second) since the configuration of traffic manager was updated (this value will wrap if no configuration changes are made for 497 days).

```
Integer getTimeLastConfigUpdate()
```

getTotalBytesIn()

Bytes received by the traffic manager from clients.

```
Long getTotalBytesIn()
```

getTotalBytesOut()

Bytes sent by the traffic manager to clients.

```
Long getTotalBytesOut()
```

getTotalConn()

Total number TCP connections received.

```
Integer getTotalConn()
```

getTotalCurrentConn()

Number of TCP connections currently established.

```
Integer getTotalCurrentConn()
```

getTrafficIPARPMessage()

Number of ARP messages sent for raised Traffic IP Addresses.

```
Integer getTrafficIPARPMessage()
```

getTrafficIPGatewayPingRequests()

Number of ping requests sent to the gateway machine.

```
Integer getTrafficIPGatewayPingRequests()
```

getTrafficIPGatewayPingResponses()

Number of ping responses received from the gateway machine.

```
Integer getTrafficIPGatewayPingResponses()
```

getTrafficIPNodePingRequests()

Number of ping requests sent to the backend nodes.

```
Integer getTrafficIPNodePingRequests()
```

getTrafficIPNodePingResponses()

Number of ping responses received from the backend nodes.

```
Integer getTrafficIPNodePingResponses()
```

getTrafficIPNumber()

The number of traffic IP addresses on this system (includes IPv4 and IPv6 addresses).

```
Integer getTrafficIPNumber()
```

getTrafficIPNumberRaised()

The number of traffic IP addresses currently raised on this system (includes IPv4 and IPv6 addresses).

```
Integer getTrafficIPNumberRaised()
```

getTrafficIPPingResponseErrors()

Number of ping response errors.

```
Integer getTrafficIPPingResponseErrors()
```

getTrafficIPState(traffic_ip_addresses) throws InvalidInput, InvalidObjectName

Whether this traffic IP address is currently being hosted by this traffic manager, for each of the specified TrafficIPs.

```
System.Stats.TrafficIPState[] getTrafficIPState(
   String[] traffic_ip_addresses
)
```

getTrafficIPTime(traffic_ip_addresses) throws InvalidInput, InvalidObjectName

The time (in hundredths of a second) since trafficIPState last changed (this value will wrap if the state hasn't changed for 497 days), for each of the specified TrafficIPs.

```
Integer[] getTrafficIPTime(
```

```
    String[] traffic_ip_addresses
)
```

getTrafficIPs()

Gets the list of Traffic IP addresses configured.

```
String[] getTrafficIPs()
```

getUniSessionCacheEntries()

The total number of universal sessions stored in the cache.

```
Integer getUniSessionCacheEntries()
```

getUniSessionCacheEntriesMax()

The maximum number of universal sessions in the cache.

```
Integer getUniSessionCacheEntriesMax()
```

getUniSessionCacheHitRate()

The percentage of universal session lookups that succeeded.

```
Integer getUniSessionCacheHitRate()
```

getUniSessionCacheHits()

Number of times a universal session entry has been successfully found in the cache.

```
Integer getUniSessionCacheHits()
```

getUniSessionCacheLookups()

Number of times a universal session entry has been looked up in the cache.

```
Integer getUniSessionCacheLookups()
```

getUniSessionCacheMisses()

Number of times a universal session entry has not been available in the cache.

```
Integer getUniSessionCacheMisses()
```

getUniSessionCacheOldest()

The age of the oldest universal session in the cache (in seconds).

```
Integer getUniSessionCacheOldest()
```

getUpTime()

The time (in hundredths of a second) that Zeus software has been operational for (this value will wrap if it has been running for more than 497 days).

```
Integer getUpTime()
```

getUserCounterNumber()

The number of user defined counters.

```
Integer getUserCounterNumber()
```

getUserCounterValue(names) throws InvalidInput, InvalidObjectName

The value of the user counter, for each of the named UserCounters.

```
Integer[] getUserCounterValue(
   String[] names
)
```

getUserCounters()

Gets the list of User counters configured.

```
String[] getUserCounters()
```

getVirtualserverBytesIn(names) throws InvalidInput, InvalidObjectName

Bytes received by this virtual server from clients, for each of the named Virtualservers.

```
Long[] getVirtualserverBytesIn(
   String[] names
)
```

getVirtualserverBytesOut(names) throws InvalidInput, InvalidObjectName

Bytes sent by this virtual server to clients, for each of the named Virtualservers.

```
Long[] getVirtualserverBytesOut(
   String[] names
)
```

getVirtualserverConnectTimedOut(names) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'connect_timeout' interval was exceeded, for each of the named Virtualservers.

```
Integer[] getVirtualserverConnectTimedOut(
   String[] names
)
```

getVirtualserverConnectionErrors(names)

Number of transaction or protocol errors in this VS, for each of the named Virtualservers.

```
Integer[] getVirtualserverConnectionErrors(
   String[] names
)
```

getVirtualserverConnectionFailures(names)

Number of connection failures in this VS, for each of the named Virtualservers.

```
Integer[] getVirtualserverConnectionFailures(
   String[] names
)
```

getVirtualserverCurrentConn(names) throws InvalidInput, InvalidObjectName

TCP connections currently established to this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverCurrentConn(
   String[] names
)
```

getVirtualserverDataTimedOut(names) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'timeout' interval was exceeded, for each of the named Virtualservers.

```
Integer[] getVirtualserverDataTimedOut(
   String[] names
)
```

getVirtualserverDirectReplies(names) throws InvalidInput, InvalidObjectName

Direct replies from this virtual server, without forwarding to a node, for each of the named Virtualservers.

```
Integer[] getVirtualserverDirectReplies(
   String[] names
)
```

getVirtualserverDiscard(names) throws InvalidInput, InvalidObjectName

Connections discarded by this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverDiscard(
   String[] names
)
```

getVirtualserverGzip(names) throws InvalidInput, InvalidObjectName

Responses which have been compressed by content compression, for each of the named Virtualservers.

```
Integer[] getVirtualserverGzip(
   String[] names
)
```

getVirtualserverGzipBytesSaved(names) throws InvalidInput, InvalidObjectName

Bytes of network traffic saved by content compression, for each of the named Virtualservers.

```
Long[] getVirtualserverGzipBytesSaved(
   String[] names
)
```

getVirtualserverHttpCacheHitRate(names) throws InvalidInput, InvalidObjectName

Percentage hit rate of the web cache for this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverHttpCacheHitRate(
   String[] names
)
```

getVirtualserverHttpCacheHits(names) throws InvalidInput, InvalidObjectName

HTTP responses sent directly from the web cache by this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverHttpCacheHits(
   String[] names
)
```

getVirtualserverHttpCacheLookups(names) throws InvalidInput, InvalidObjectName

HTTP requests that are looked up in the web cache by this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverHttpCacheLookups(
   String[] names
)
```

getVirtualserverHttpRewriteCookie(names) throws InvalidInput, InvalidObjectName

HTTP Set-Cookie headers, supplied by a node, that have been rewritten, for each of the named Virtualservers.

```
Integer[] getVirtualserverHttpRewriteCookie(
   String[] names
)
```

getVirtualserverHttpRewriteLocation(names) throws InvalidInput, InvalidObjectName

HTTP Location headers, supplied by a node, that have been rewritten, for each of the named Virtualservers.

```
Integer[] getVirtualserverHttpRewriteLocation(
   String[] names
)
```

getVirtualserverKeepaliveTimedOut(names) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'keepalive_timeout' interval was exceeded, for each of the named Virtualservers.

```
Integer[] getVirtualserverKeepaliveTimedOut(
   String[] names
```

```
)
```

getVirtualserverMaxConn(names) throws InvalidInput, InvalidObjectName

Maximum number of simultaneous TCP connections this virtual server has processed at any one time, for each of the named Virtualservers.

```
Integer[] getVirtualserverMaxConn(
   String[] names
)
```

getVirtualserverNumber()

The number of virtual servers.

```
Integer getVirtualserverNumber()
```

getVirtualserverPort(names) throws InvalidInput, InvalidObjectName

The port the virtual server listens on, for each of the named Virtualservers.

```
Integer[] getVirtualserverPort(
   String[] names
)
```

getVirtualserverProtocol(names) throws InvalidInput, InvalidObjectName

The protocol the virtual server is operating, for each of the named Virtualservers.

```
System.Stats.VirtualserverProtocol[] getVirtualserverProtocol(
   String[] names
)
```

getVirtualserverSIPRejectedRequests(names) throws InvalidInput, InvalidObjectName

Number of SIP requests rejected due to them exceeding the maximum amount of memory allocated to the connection, for each of the named Virtualservers.

```
Integer[] getVirtualserverSIPRejectedRequests(
   String[] names
)
```

getVirtualserverSIPTotalCalls(names) throws InvalidInput, InvalidObjectName

Total number of SIP INVITE requests seen by this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverSIPTotalCalls(
   String[] names
)
```

getVirtualserverTotalConn(names) throws InvalidInput, InvalidObjectName

Requests received by this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverTotalConn(
   String[] names
)
```

getVirtualserverTotalDgram(names) throws InvalidInput, InvalidObjectName

UDP datagrams processed by this virtual server, for each of the named Virtualservers.

```
Integer[] getVirtualserverTotalDgram(
   String[] names
)
```

getVirtualserverUdpTimedOut(names) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'udp_timeout' interval was exceeded, for each of the named Virtualservers.

```
Integer[] getVirtualserverUdpTimedOut(
   String[] names
)
```

getVirtualservers()

Gets the list of Virtual Servers configured and enabled.

```
String[] getVirtualservers()
```

getWebCacheEntries()

The number of items in the web cache.

```
Integer getWebCacheEntries()
```

getWebCacheHitRate()

The percentage of web cache lookups that succeeded.

```
Integer getWebCacheHitRate()
```

getWebCacheHits()

Number of times a page has been successfully found in the web cache.

```
Long getWebCacheHits()
```

getWebCacheLookups()

Number of times a page has been looked up in the web cache.

```
Long getWebCacheLookups()
```

getWebCacheMaxEntries()

The maximum number of items in the web cache.

```
Integer getWebCacheMaxEntries()
```

getWebCacheMemMaximum()

The maximum amount of memory the web cache can use in kilobytes.

```
Integer getWebCacheMemMaximum()
```

getWebCacheMemUsed()

Total memory used by the web cache in kilobytes.

```
Integer getWebCacheMemUsed()
```

getWebCacheMisses()

Number of times a page has not been found in the web cache.

```
Long getWebCacheMisses()
```

getWebCacheOldest()

The age of the oldest item in the web cache (in seconds).

```
Integer getWebCacheOldest()
```

getZxtmNumber()

The number of traffic managers in the cluster.

```
Integer getZxtmNumber()
```

6.30.2 Structures

System.Stats.Node

Represents a Node object.

```
struct System.Stats.Node {
   # The IPv4 or IPv6 address of this node.
   String Address;

   # The port this node listens on.
   Integer Port;
}
```

System.Stats.PerNodeServiceLevel

Represents a PerNodeServiceLevel object.

```
struct System.Stats.PerNodeServiceLevel {
   # The name of the SLM class.
   String SLMName;

   # The IP address of this node.
   String NodeAddress;

   # The port number of this node.
   Integer NodePort;
}
```

System.Stats.PerPoolNode

Represents a PerPoolNode object.

```
struct System.Stats.PerPoolNode {
   # The name of the pool that this node belongs to.
   String PoolName;

   # The IPv4 or IPv6 address of this node.
   String NodeAddress;

   # The port that this node listens on.
   Integer NodePort;
}
```

6.30.3 Enumerations

System.Stats.NodeState

```
enum System.Stats.NodeState {
   alive,
```

```
   dead,

   unknown
}
```

System.Stats.PerPoolNodeState

```
enum System.Stats.PerPoolNodeState {
   alive,

   dead,

   unknown,

   draining
}
```

System.Stats.PoolAlgorithm

```
enum System.Stats.PoolAlgorithm {
   roundrobin,

   weightedRoundRobin,

   perceptive,

   leastConnections,

   fastestResponseTime,

   random,

   weightedLeastConnections
}
```

System.Stats.PoolPersistence

```
enum System.Stats.PoolPersistence {
   none,

   ip,

   rule,

   transparent,

   applicationCookie,
```

```
   xZeusBackend,

   ssl
}
```

System.Stats.PoolState

```
enum System.Stats.PoolState {
   active,

   disabled,

   draining,

   unused,

   unknown
}
```

System.Stats.ServiceLevelIsOK

```
enum System.Stats.ServiceLevelIsOK {
   notok,

   ok
}
```

System.Stats.TrafficIPState

```
enum System.Stats.TrafficIPState {
   raised,

   lowered
}
```

System.Stats.VirtualserverProtocol

```
enum System.Stats.VirtualserverProtocol {
   http,

   https,

   ftp,

   imaps,

   imapv2,

   imapv3,
```

```
    imapv4,
    pop3,
    pop3s,
    smtp,
    ldap,
    ldaps,
    telnet,
    sslforwarding,
    udpstreaming,
    udp,
    dns,
    genericserverfirst,
    genericclientfirst,
    dnstcp,
    sipudp,
    siptcp,
    rtsp
}
```

6.31 System.Management

URI: `http://soap.zeus.com/zxtm/1.0/System/Management/`

The System.Management interface provides methods to manage the traffic manager and the system, such as restarting the software.

6.31.1 Methods

rebootSystem()

Perform a system reboot.

```
void rebootSystem()
```

restartAFM() throws InvalidOperation

Restart the Zeus Application Firewall Module on all machines. Any connections currently using Zeus AFM will be aborted.

```
void restartAFM()
```

restartJava()

Restart the Java Extension support. Any connections currently using a Java Extension will be aborted.

```
void restartJava()
```

restartTrafficManager()

Restarts the traffic manager software. Any connections currently being handled will be aborted.

```
void restartTrafficManager()
```

shutdownSystem()

Perform a system shutdown.

```
void shutdownSystem()
```

6.32 AFM

URI: `http://soap.zeus.com/zxtm/1.0/AFM/`

The AFM interface allows management of the Zeus Application Firewall Module.

6.32.1 Methods

getAdminMasterPort() throws InvalidOperation

Get the Zeus AFM XML Master port, this port is used on all IP addresses.

```
Unsigned Integer getAdminMasterPort()
```

getAdminServerPort() throws InvalidOperation

Get the Zeus AFM Administration server port, this port is used on all IP addresses.

```
Unsigned Integer getAdminServerPort()
```

getAdminServerSessionTimeout() throws InvalidOperation

Get the period of inactivity after which Zeus AFM Administration Server sessions will automatically timeout.

```
Unsigned Integer getAdminServerSessionTimeout()
```

getAdminSlavePort() throws InvalidOperation

Get the Zeus AFM XML Slave port, this port is used on all IP addresses.

```
Unsigned Integer getAdminSlavePort()
```

getClusterState()

Get state data for the Zeus Application Firewall Module across all machines in the cluster.

```
AFM.State[] getClusterState()
```

getDeciderBasePort() throws InvalidOperation

Get the Zeus AFM base decider port. This port, plus one port per decider process above this port, will be used for the Zeus AFM deciders. For example, if set to 8100 then ports 8100, 8101, and 8102 will be used.

```
Unsigned Integer getDeciderBasePort()
```

getInternalDeciderBasePort() throws InvalidOperation

Get the Zeus AFM internal decider communication base port. Zeus AFM requires ports for internal communication, these ports are bound to localhost (127.0.0.1) only. This sets the base for these communication ports, when Zeus AFM is started it will start at this port and work its way up taking available ports until it has enough ports.

```
Unsigned Integer getInternalDeciderBasePort()
```

getNumberOfDeciders() throws InvalidOperation

Get the number of Zeus AFM decider processes to run.

```
Unsigned Integer getNumberOfDeciders()
```

getVersion()

Get the version of the Zeus Application Firewall Module installed on the traffic manager. Returns an empty string if Zeus AFM is not installed.

```
String getVersion()
```

setAdminMasterPort(value) throws InvalidOperation, InvalidInput

Set the Zeus AFM XML Master port, this port is used on all IP addresses.

```
void setAdminMasterPort(
   Unsigned Integer value
)
```

setAdminServerPort(value) throws InvalidOperation, InvalidInput

Set the Zeus AFM Administration server port, this port is used on all IP addresses.

```
void setAdminServerPort(
   Unsigned Integer value
)
```

setAdminServerSessionTimeout(value) throws InvalidOperation, InvalidInput

Set the period of inactivity after which Zeus AFM Administration Server sessions will automatically timeout.

```
void setAdminServerSessionTimeout(
   Unsigned Integer value
)
```

setAdminSlavePort(value) throws InvalidOperation, InvalidInput

Set the Zeus AFM XML Slave port, this port is used on all IP addresses.

```
void setAdminSlavePort(
   Unsigned Integer value
)
```

setDeciderBasePort(value) throws InvalidOperation, InvalidInput

Set the Zeus AFM base decider port. This port, plus one port per decider process above this port, will be used for the Zeus AFM deciders. For example, if set to 8100 then ports 8100, 8101, and 8102 will be used.

```
void setDeciderBasePort(
   Unsigned Integer value
)
```

setInternalDeciderBasePort(value) throws InvalidOperation, InvalidInput

Set the Zeus AFM internal decider communication base port. Zeus AFM requires ports for internal communication, these ports are bound to localhost (127.0.0.1) only. This sets the base for these communication ports, when Zeus AFM is started it will start at this port and work its way up taking available ports until it has enough ports.

```
void setInternalDeciderBasePort(
   Unsigned Integer value
)
```

setNumberOfDeciders(value) throws InvalidOperation, InvalidInput

Set the number of Zeus AFM decider processes to run.

```
void setNumberOfDeciders(
   Unsigned Integer value
)
```

uninstall() throws InvalidOperation

Uninstalls the Zeus Application Firewall Module on the traffic manager.

```
void uninstall()
```

6.32.2 Structures

AFM.BasicStatus

Contains basic Zeus AFM runtime status information.

```
struct AFM.BasicStatus {
   # Whether or not Zeus AFM is installed.
   String installed;

   # Whether or not Zeus AFM is running.
   String running;

   # The version of Zeus AFM installed.
   String version;

   # Whether or not the maching is clustered with the local Zeus
   # AFM.
   String clustered;
}
```

AFM.ClusterStatus

Contains a Zeus AFM state message.

```
struct AFM.ClusterStatus {
   # Cluster member this status is for.
   String member;

   # Status of the cluster member.
   String status;
```

```
}
```

AFM.State

Contains status information about a Zeus AFM installation.

```
struct AFM.State {
   # Name of the machine this information is from.
   String machine;

   # Describes the basic runtime status of Zeus AFM on a machine.
   AFM.BasicStatus basicstatus;

   # State messages from the Zeus AFM on the machine.
   AFM.StateMessage[] messages;

   # Statuses for all AFMs in the AFM's cluster.
   AFM.ClusterStatus[] messages;

   # Strings describing any general errors relating to Zeus AFM.
   String[] errors;
}
```

AFM.StateMessage

Contains a Zeus AFM state message.

```
struct AFM.StateMessage {
   # State for this message, either OK or ERROR.
   String state;

   # Message describing the reason for the state.
   String message;
}
```

6.33 SOAP Faults

When a function encounters an error it will emit a fault. Depending on the fault that occurred the fault structure will contain more information related to the fault. The documentation for individual functions lists the different types of faults that a function can emit.

6.33.1 Faults

DeploymentError

The DeploymentError fault is raised when a configuration change causes errors when attempting to apply the configuration to a running traffic manager. It would be raised in cases such as failing to bind to a port when enabling a Virtual Server.

```
struct DeploymentError {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault (if
   # appropriate)
   String object;

   # The configuration key that caused the fault (if appropriate)
   String key;

   # The value that caused the fault (if appropriate)
   String value;
}
```

InvalidInput

The InvalidInput fault is raised when the input to a function is invalid, for example a number was out of range. This fault is also raised in cases such as VirtualServer.setPool() where the Pool doesn't exist The details in the fault contain the object, key and value that caused the fault. These might be blank if they are not relevant to the fault.

```
struct InvalidInput {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault (if
   # appropriate)
   String object;

   # The configuration key that caused the fault (if appropriate)
   String key;

   # The value that caused the fault (if appropriate)
   String value;
}
```

InvalidObjectName

The InvalidObjectName fault is raised when attempting to create a new object (e.g. via an add, rename or copy) and the name is invalid (e.g. it contains a '/').

```
struct InvalidObjectName {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault
   String object;
}
```

InvalidOperation

The InvalidOperation fault is emitted when attempting an operation that doesn't make sense or is prohibited, for example deleting a built-in monitor, or attempting to rename an object twice in the same call.

```
struct InvalidOperation {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault (if
   # appropriate)
   String object;

   # The configuration key that caused the fault (if appropriate)
   String key;

   # The value that caused the fault (if appropriate)
   String value;
}
```

LicenseError

The LicenseError fault is emitted when attempting to use functionality that is disabled by the license key. You will need to contact your support provider to get a new license key with the required functionality. There may be a charge for this.

```
struct LicenseError {
   # A human readable string describing the error
   String errmsg;

   # The license key feature that was missing
   String feature;
}
```

ObjectAlreadyExists

The ObjectAlreadyExists fault is raised when attempting to create an object (such as a Virtual Server) that already exists. It will also be raised in cases such as renaming and copying objects.

```
struct ObjectAlreadyExists {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault
   String object;
}
```

ObjectDoesNotExist

The ObjectDoesNotExist fault is raised when attempting to perform an operation on an object (such as Virtual Server) that doesn't exist. This fault will only be raised if the primary object in the call doesn't exist. For example if calling VirtualServer.setPool(), then this fault will be raised if the Virtual Server doesn't exist, but if the Pool doesn't exist then the "InvalidInput" fault will be raised.

```
struct ObjectDoesNotExist {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault
   String object;
}
```

ObjectInUse

The ObjectInUse fault is raised when attempting to delete an object that is referenced by another object, for example deleting a Pool that is in use by a Virtual Server.

```
struct ObjectInUse {
   # A human readable string describing the error
   String errmsg;

   # The name of the object that caused the fault
   String object;
}
```

Further Information

7.1 Zeus Manuals

Bundled with the software is a Getting Started Guide, intended to get you up and running quickly with the software. If you have purchased a 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 is also a full User Guide, and a manual for the TrafficScript rules language.

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/.

7.2 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