Brocade Virtual Traffic Manager: Control API Guide

Supporting 17.2
# Contents

**Preface**
- Document Conventions: 1
- Notes and Warnings: 1
- Text Formatting Conventions: 2
- Command Syntax Conventions: 2

**Brocade Resources**: 3

**Document Feedback**: 3

**Contacting Brocade Technical Support**
- Brocade Customers: 3
- Brocade OEM Customers: 4

**Chapter 1 - Introduction**
- About This Guide: 5
- Introducing the Control API: 5
- Standards-Conformant SOAP Communications: 5
- A SOAP-Based Architecture: 6
- Security Considerations: 6

**Chapter 2 - Code Samples**
- Listing Running Virtual Servers: 9
  - listVS.pl using Perl SOAP::Lite: 9
  - listVS.cs using C Sharp: 10
  - listVS.java using Java: 12
  - listVS.py using Python: 14
  - listVS.php using PHP 5: 14

- Fault Handling: 15
  - Fault Handling with SOAP::Lite: 16
  - Fault Handling using C Sharp: 16
  - Fault Handling using Java: 17
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog.Authenticators</td>
<td>316</td>
</tr>
<tr>
<td>Catalog.DNSServer.ZoneFiles</td>
<td>324</td>
</tr>
<tr>
<td>Catalog.JavaExtension</td>
<td>313</td>
</tr>
<tr>
<td>Catalog.Monitor</td>
<td>247</td>
</tr>
<tr>
<td>Catalog.Protection</td>
<td>280</td>
</tr>
<tr>
<td>Catalog.Rate</td>
<td>311</td>
</tr>
<tr>
<td>Catalog.SLM</td>
<td>307</td>
</tr>
<tr>
<td>Catalog.SSL.Certificates</td>
<td>269</td>
</tr>
<tr>
<td>Catalog.SSL.CertificateAuthorities</td>
<td>273</td>
</tr>
<tr>
<td>Catalog.SSL.CertificateAuthorities</td>
<td>273</td>
</tr>
<tr>
<td>Catalog.SSL.ClientCertificates</td>
<td>276</td>
</tr>
<tr>
<td>Catalog.SSL.DNSSEC</td>
<td>279</td>
</tr>
<tr>
<td>Catalog.Bandwidth</td>
<td>304</td>
</tr>
<tr>
<td>Catalog.DNSServer.ZoneFiles</td>
<td>324</td>
</tr>
<tr>
<td>Enumerations</td>
<td>524</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>System.MachineInfo</td>
<td>528</td>
</tr>
<tr>
<td>Methods</td>
<td>528</td>
</tr>
<tr>
<td>Structures</td>
<td>529</td>
</tr>
<tr>
<td>System.NAT</td>
<td>529</td>
</tr>
<tr>
<td>Methods</td>
<td>529</td>
</tr>
<tr>
<td>Structures</td>
<td>531</td>
</tr>
<tr>
<td>System.RequestLogs</td>
<td>532</td>
</tr>
<tr>
<td>Methods</td>
<td>532</td>
</tr>
<tr>
<td>Structures</td>
<td>533</td>
</tr>
<tr>
<td>System.Stats</td>
<td>533</td>
</tr>
<tr>
<td>Methods</td>
<td>533</td>
</tr>
<tr>
<td>Structures Enumerations</td>
<td>577</td>
</tr>
<tr>
<td>System.Management</td>
<td>581</td>
</tr>
<tr>
<td>Methods</td>
<td>581</td>
</tr>
<tr>
<td>AFM</td>
<td>581</td>
</tr>
<tr>
<td>Methods</td>
<td>582</td>
</tr>
<tr>
<td>Structures</td>
<td>585</td>
</tr>
<tr>
<td>Location</td>
<td>586</td>
</tr>
<tr>
<td>Methods</td>
<td>586</td>
</tr>
<tr>
<td>Structures</td>
<td>588</td>
</tr>
<tr>
<td>Users</td>
<td>589</td>
</tr>
<tr>
<td>Methods</td>
<td>589</td>
</tr>
<tr>
<td>GLB.Service</td>
<td>590</td>
</tr>
<tr>
<td>Methods</td>
<td>590</td>
</tr>
<tr>
<td>Structures</td>
<td>603</td>
</tr>
<tr>
<td>Enumerations</td>
<td>604</td>
</tr>
<tr>
<td>System.CloudCredentials</td>
<td>605</td>
</tr>
<tr>
<td>Methods</td>
<td>605</td>
</tr>
<tr>
<td>Structures</td>
<td>610</td>
</tr>
<tr>
<td>System.Steelhead</td>
<td>610</td>
</tr>
<tr>
<td>Methods</td>
<td>610</td>
</tr>
<tr>
<td>Enumerations</td>
<td>614</td>
</tr>
<tr>
<td>Catalog.Aptimizer.Profile</td>
<td>615</td>
</tr>
<tr>
<td>Methods</td>
<td>615</td>
</tr>
<tr>
<td>Enumerations</td>
<td>619</td>
</tr>
<tr>
<td>Catalog.Kerberos.Principals</td>
<td>619</td>
</tr>
<tr>
<td>Methods</td>
<td>619</td>
</tr>
<tr>
<td>Structures</td>
<td>624</td>
</tr>
<tr>
<td>Catalog.Kerberos.KeyTabs</td>
<td>624</td>
</tr>
<tr>
<td>Methods</td>
<td>625</td>
</tr>
<tr>
<td>Catalog.Kerberos.Krb5Conf</td>
<td>625</td>
</tr>
<tr>
<td>Methods</td>
<td>625</td>
</tr>
</tbody>
</table>
Preface

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

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

Document Conventions

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

Notes and Warnings

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

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

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

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

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

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bold text</strong></td>
<td>Identifies command names</td>
</tr>
<tr>
<td></td>
<td>Identifies keywords and operands</td>
</tr>
<tr>
<td></td>
<td>Identifies the names of user-manipulated GUI elements</td>
</tr>
<tr>
<td></td>
<td>Identifies text to enter at the GUI</td>
</tr>
<tr>
<td><em>italic text</em></td>
<td>Identifies emphasis</td>
</tr>
<tr>
<td></td>
<td>Identifies variables</td>
</tr>
<tr>
<td></td>
<td>Identifies document titles</td>
</tr>
<tr>
<td><strong>Courier font</strong></td>
<td>Identifies CLI output</td>
</tr>
<tr>
<td></td>
<td>Identifies command syntax examples</td>
</tr>
</tbody>
</table>

Command Syntax Conventions

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

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bold text</strong></td>
<td>Identifies command names, keywords, and command options.</td>
</tr>
<tr>
<td><em>italic text</em></td>
<td>Identifies a variable.</td>
</tr>
<tr>
<td>value</td>
<td>In Fibre Channel products, a fixed value provided as input to a command option is printed in plain text. For example, –show WWN.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.</td>
</tr>
<tr>
<td>{ x</td>
<td>y</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Nonprinting characters, for example, passwords, are enclosed in angle brackets.</td>
</tr>
<tr>
<td>...</td>
<td>Repeat the previous element, for example, member[member...].</td>
</tr>
<tr>
<td>\</td>
<td>Indicates a “soft” line break in command examples. If a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.</td>
</tr>
</tbody>
</table>
Brocade Resources

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

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

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

Document Feedback

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

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

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

Contacting Brocade Technical Support

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

Brocade Customers

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

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

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

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

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

- “About This Guide,” next
- “Introducing the Control API” on page 5
- “A SOAP-Based Architecture” on page 6

### About This Guide

The *Brocade Virtual Traffic Manager: Control API Guide* describes the Traffic Manager SOAP-based Control API.

This guide introduces you to the syntax and constructs used in the Traffic Manager’s Control API, and is intended as a complete reference to all SOAP object types and methods available in the Traffic Manager.

### Introducing the Control API

A cluster of Traffic Managers is normally managed using the Web-based Admin UI on one of the cluster members.

The Traffic Manager’s Control API provides an alternative means to remotely administer and configure a Traffic Manager 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. After it had provisioned additional servers, it could then reconfigure the server pools on your Traffic Managers using the Control API.

### 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. The Traffic Manager’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 Traffic Manager’s WSDL files are located in \texttt{ZEUSHOME/zxtm/etc/wsdl}. You can download the WSDL files from the SOAP API page in the Traffic Manager’s Online Help system.

A SOAP-compliant programming environment parses the WSDL files to determine which remote methods can be called, and then allows 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, and so on. The application developer can concentrate on implementing the control logic required to support the application they are building.

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

A SOAP-Based Architecture

Figure 1-1. Arrangement of Management Server, Traffic Manager Cluster and Server Nodes

A management application can issue a SOAP request to one of the Traffic Managers in a cluster. The application might be running on a stand alone management server, one of the server nodes, or on one of the Traffic Managers themselves.

The application can issue the request to any of the Traffic Managers in a cluster. All Traffic Managers then automatically synchronize their configuration, so a configuration change sent to one Traffic Manager is automatically replicated across the cluster.

Security Considerations

The SOAP-based management application communicates with a SOAP server running on the Traffic Manager’s Admin Server (the Traffic Manager-based service used to provide the Administration UI), 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 Admin Server authenticates itself with its SSL certificate, which is typically self-signed.
- You might need to ensure that your SOAP application accepts self-signed certificates, or install a trusted SSL certificate in your 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 Admin Server. To define a group, click **System > Users > Groups** in the Traffic Manager Admin UI.

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

You might want to define a specific username for your management application to use so that you can track its activity using Traffic Manager’s **Audit Log**.
CHAPTER 2  
Code Samples

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

This chapter contains the following sections:
- “Listing Running Virtual Servers,” next
- “Fault Handling” on page 15

Listing Running Virtual 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 “SOAP Control API” permissions (see “Security Considerations” on page 6).
- 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 latest version of the VirtualServer interface, generally with reference to the WSDL specification.
- 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.

listVS.pl using Perl SOAP::Lite

#!/usr/bin/perl -w

1. Some environments, such as Perl’s SOAP::Lite do not validate method calls against the WSDL specification.
Code Samples

Listing Running Virtual Servers

use SOAP::Lite 0.60;
# This is the url of the Traffic Manager 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");
# 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 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



Sites that use CPAN can obtain the modules from the following URLs:

rpms.

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 version of 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 are
detected at runtime and SOAP structures and enumerations must be managed manually (for further
information, see Chapter 3, “Using Perl SOAP::Lite” ). The SSL layer is able to accept self-signed certificates.

listVS.cs using C Sharp
using
using
using
using

System;
System.Net;
System.IO;

public class AllowSelfSignedCerts : IcertificatePolicy {
public bool CheckValidationResult( ServicePoint sp,

10

Brocade Virtual Traffic Manager: Control API Guide


Listing Running Virtual Servers

```csharp
X509Certificate cert, WebRequest request, int problem )
{
    return true;
}
}

public class listVS {

    public static void Main( string[] args )
    {
            new AllowSelfSignedCerts();

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

        try {
            VirtualServer p = new 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 Mono1.

Using .Net 1.1, compile and run the example code as follows:
C:\> wsdl -o:VirtualServer.cs -n:Stingray VirtualServer.wsdl
C:\> csc /out:listVS.exe VirtualServer.cs listVS.cs
C:\> listVS.exe

Main website
Mail servers
Test site

Using Mono, compile and run as follows:
$ wsdl -o:VirtualServer.cs -n:Stingray VirtualServer.wsdl
$ mcs /out:listVS.exe /r:System.Web.Services \VirtualServer.cs listVS.cs
$ listVS.exe
Main website
Mail servers
Test site

To obtain the WSDL interface specifications for the Control API, use the files located on the Traffic Manager file system in ZEUSHOME/zxtm/etc/wsdl/. Alternatively, download them from the “SOAP API” page of the Traffic Manager’s Online Help.

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

1. Use the most recent build of Mono from .
listVS.java using Java

```java
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()
                };
            }
        }
    }
}
```

| Brocade Virtual Traffic Manager: Control API Guide | Code Samples | Listing Running Virtual Servers |
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 majority of this code disables client certificate checking. Details of the code and surrounding infrastructure are available at:

This code works with the Java 1.5 SDK/JRE.

**To build and run the code**

1. Obtain a WSDL-to-Java converter by downloading “axis” from the Apache Web site: Copy all the .jar files from axis-<version>/libs/ to the JAVAHOME/jre/lib/ext/ directory, or add them to your CLASSPATH.

2. To avoid warnings when the code is run, use the following URLs to download and install the Java Activation Framework and JavaMail libraries.
   - For Java Activation Framework:

   Copy activation.jar and mail.jar from these packages to JAVAHOME/jre/lib/ext/, or add them to your CLASSPATH.

3. From your build directory, type the following command to convert the required WSDL files into Java code1:

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

4. To compile and run the example, type the following commands:

   javac listVS.java
   java listVS

   This should produce the following output:

   Main website
   Mail servers
   Test site

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

1. The WSDL interface specifications for the Control API are located in ZEUSHOME/zxtm/etc/wsd1/.
For example, if you want to access functions within the “XXX” interface, you need to instantiate an XXXLocator object, declare the location of the Traffic Manager using the function setXXXPortEndpointAddress() 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:

```java
SystemCacheLocator scl = new SystemCacheLocator();
scl.setSystemCachePortEndpointAddress("https://username:password@host:9090/soap");
SystemCachePort scp = scl.getSystemCachePort();
/* Invoke the methods on the SystemCachePort object */
scp.clearWebCache();
```

### listVS.py using Python

```
#!/usr/bin/python

import SOAPpy

c = SOAPpy.WSDL.Proxy("VirtualServer.wsdl")
names = c.getVirtualServerNames()
enabled = c.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. However, for security reasons, the location of the Admin Server (including the required administrator username and password) is not embedded in the Traffic Manager 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. Locate the “soap:address” node at the end of each WSDL file and edit appropriately:

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

Run the Python script:

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

### listVS.php using PHP 5

```
#!/usr/bin/php

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

1. The WSDL interface specifications for the Control API are located in `ZEUSHOME/zxtm/etc/wsdl/`.
2. 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.
```
Fault Handling

The 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 example, an “ObjectDoesNotExist” fault is returned when trying to set a property for a Virtual Server that doesn’t exist. Information contained inside the fault helps to 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. Refer to your API documentation for details on how best to handle faults.

Code Samples

```php
<?php
$names = $conn->getVirtualServerNames();
$enabled = $conn->getEnabled($names);

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

You might need to enable the SOAP extensions in your `php.ini` file. To do this, follow the instructions at ...

By default, the PHP Soap toolkit expects to find the location of the SOAP server in the WSDL file. To override this behavior, use the method `__setLocation()`. For example:

```php
#!/usr/bin/php
<?php
$conn = new SoapClient( "VirtualServer.wsdl",
    array('login' => "username", 'password' => "password")
);
$conn->__setLocation('https://host:9090/soap');
```

Alternatively, load the WSDL interactively from the Traffic Manager:

```php
<?php
$stm_url = "https://host:9090";

$conn = new SoapClient(
    $stm_url. '/apps/zxtm/wsdl/VirtualServer.wsdl',
    array('login' => "username", 'password' => "password")
);
$conn->__setLocation($stm_url.'/soap');
```

For more details on `__setLocation()`, see ...

You can also specify the details from your application, so these do not need to be embedded in the WSDL:

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

Run the PHP script as follows:

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

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

**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:

```perl
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 occurred
  }
  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 ] );
```

**Fault Handling using C Sharp**

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

```csharp
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 );

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 );
}
CHAPTER 3

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.

This chapter contains the following sections:
- “Control API Methods,” next
- “Control API Enumerations” on page 20
- “Control API Structures” on page 21

Control API Methods

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

```perl
# 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), construct a SOAP::Lite connection object for each one.

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

**Note:** When a new version of the Traffic Manager is released, individual interfaces might be changed in some way that alters their use or behavior. This could be in order to fix a bug or to provide additional functionality. If this situation occurs, instead of amending the existing interface, a new later version-numbered interface is included with the release (which Brocade recommends the use of). However, previous versions are always preserved to ensure backwards compatibility with your existing applications. The interface version is identified in the “ns” string provided to the connection object, and the function reference listing contained in this document always refers to the latest version available.
**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:

```plaintext
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.

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

- `setLoadBalancingAlgorithm( names, values )`  
  Set the load balancing algorithms that each of the named pools uses.

```plaintext
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 context:

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

In a SOAP response, you must provide a custom `Deserializer` so that the SOAP::Lite library can convert the values in the SOAP response into appropriate internal representations (in other words, literal strings):

```perl
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;
    };
}
```

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

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

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:

```
# 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 {
    # 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.
    ...
Using Perl SOAP::Lite

# 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 (the protocol: VirtualServer.Protocol) and a string (the name of the default pool).

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

```perl
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:

```perl
$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:

```perl
#!/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 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:
   port $r->{port}, protocol $r->{protocol}, " .
   'pool $r->{default_pool}\n";
CHAPTER 4  Sample Control API Applications

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

This chapter contains the following sections:

- “Blocking Traffic from an IP Address,” next
- “Adding a Node to a Pool” on page 27
- “Reconfiguring Your Site Based on Traffic Load” on page 30

## Blocking Traffic from an IP Address

An Intrusion Detection System (IDS) or a live log analysis tool might 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 Manager. 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 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.

### Perl Example

```perl
#!/usr/bin/perl -w
use SOAP::Lite 0.60;

# This is the url of the Traffic Manager 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
```
Sample Control API Applications

Blocking Traffic from an IP Address

```perl
-> 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**

This code sample accesses the “/Catalog/Protection” URI to edit a Service Protection class. With a WSDL-based interface, you instead use the “Catalog.Protection.wsdl” interface.

The sample then uses the `addBannedAddresses()` function with 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 also includes a basic `on_fault` handler, called if an error occurs. The handler reports a transport error if the SOAP application could not connect to the remote SOAP server. Otherwise, it reports a SOAP error.

For more a more sophisticated example of a Perl fault handler, see “Fault Handling with SOAP::Lite” on page 16.

**C# Example**

```csharp
using System;
using System.Net;
using System.IO;

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 )
  {
      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 {
      CatalogProtection p = new CatalogProtection();
      p.Url = url;
      p.Credentials = new NetworkCredential(
```

26  Brocade Virtual Traffic Manager: Control API Guide
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, load is slowly ramped up on newly introduced nodes in order to gauge their potential performance. This continues 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.

Perl Example

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

use SOAP::Lite 0.60;

# This is the url of the Traffic Manager 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}} ) {
    p.addBannedAddresses( new string[] { name },
        new string[][] { new string[] { badIP } } );
} catch ( Exception e ) {
    Console.WriteLine( "{0}", e );
}
```

Sample Control API Applications

Adding a Node to a Pool

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 does not exist, a SOAP fault is raised. Perl’s “on_fault” handler is 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 Control API methods are all bulk-enabled. In other words, they are designed to process lists of objects efficiently.

The listVS example can 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 Control API application can update the configuration by modifying the hash:

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

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

$res = $conn->setEnabled
    [ keys %enabled ], [ values %enabled ]

C# Example

using System;
using System.Net;
using System.IO;

```csharp
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 )
    {
            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 {
            Pool p = new 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 );
        }
    }
}
```
Reconfiguring Your Site Based on Traffic Load

This code example monitors the performance of the JSP pages on a Web site. If the performance drops below an acceptable level, the example uses the Control API to enable a TrafficScript rule that prevents more users from logging in to the Web site. After performance climbs back to an acceptable level, the Control API application can disable the rule.

The TrafficScript Rule

To prevent users from logging in to the Web site, use a TrafficScript rule similar to the following:

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

Add this rule to the Rules catalog and name it “Disable Login” (referenced later). Configure it as a request rule for your Virtual Server, but set it to be disabled. The Control API application uses 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.

Perl Example

```perl
#!/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 ) = 0_;
        ...
    }

    # Other code...
```
if( $type && $type =~ m@http://soap.zeus.com/zxtm/0/@) {
    return $val;
} return undef;
};

# This is the url of the Traffic Manager 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; })
    -> timeout(10);

# Get a list of rules used by the named virtual server
my $res = $conn->getRules( [ $vsName ]);
my @rules = @{$res->[0]->@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 rule’s 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 VirtualServer.Rule containing an enumeration of type VirtualServerRuleRunFlag.

C# Example
using System;
using System.Net;
using System.IO;

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 )
    {
            new AllowSelfSignedCerts();

        string url = "https://host:9090/soap";
        string username = "username";
        string password = "password";
        string vsName = "Main web site";
        string rule = "Disable login";
        try {
            VirtualServer p = new VirtualServer();
            p.Url = url;
            p.Credentials = new NetworkCredential(
                username, password);

            // Get a list of rules used by the named
            // virtual server
            VirtualServerRule[][] rules =
                p.getRules( new string[] { vsName } );

            // Search for the rule to enable
            bool found = false;
            foreach ( 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
                VirtualServerRule[] newrules = new
                    VirtualServerRule[ rules[0].Length + 1 ];
                newrules[0] = new VirtualServerRule();
                newrules[0].name = rule;
                newrules[0].enabled = true;
                newrules[0].run_frequency =
                    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 );
        }
    }
}
CHAPTER 5 Troubleshooting

This chapter lists some useful troubleshooting techniques that you can use when building Control API applications.

This chapter contains the following sections:

- “General Debugging Techniques,” next
- “Debugging with Perl” on page 34
- “Debugging with C#” on page 35
- “Debugging with Java” on page 36

General Debugging Techniques

File Locations

The WSDL interface specifications are located in `ZEUSHOME/zxtm/etc/wsdl/`.

`ZEUSHOME` is the installation directory for your Traffic Manager software, typically `/usr/local/zeus` or `/opt/zeus` depending on your product variant.

Alternatively, download the WSDL files from the SOAP API page in the Online Help.

Log Files

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

- Traffic Manager core software: `ZEUSHOME/log/errors`
  Validation errors and incorrect configuration problems are reported in this log file.

- Admin Server and Admin UI: `ZEUSHOME/admin/log/errors`
  The Admin Server processes the SOAP requests and sends the new configuration to the Traffic Manager software. Any SOAP protocol or transport errors are reported here.
Troubleshooting

Snooping the SOAP Traffic

A Control API application sends SOAP requests to the Traffic Manager Admin Server, which typically listens using SSL on port 9090. If your SOAP toolkit does not support debugging or tracing, use a network snooping tool such as WireShark\(^1\) 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.

If your SOAP transaction is encrypted with SSL, to enable inspection you must first disable SSL on the Admin Server by performing the following steps:

1. Edit the file $ZEUSHOME/admin/website and comment out the line “security!enabled yes”. To do this, prefix the line with a hash (#), as per the following:

   # security!enabled yes

2. Restart the Traffic Manager software by typing the following command\(^2\):

   $ZEUSHOME/restart-zeus

   ($ZEUSHOME is the Traffic Manager software installation directory for your product variant.)

3. Modify your Control API application to use an “http://” URL rather than an “https://” URL.

Debugging with Perl

Problems with WSDL Interfaces

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

Ensure that any URIs used in the SOAP::Lite objects you construct in your application are correct. To confirm correct URI usage, refer to the reference information in Chapter 6, “Function Reference.”

For example, to reference methods in the VirtualServer interface, 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/

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.

For further information, see “Fault Handling” on page 15.

1. WireShark is available for all major Operating Systems. For further information, see

2. Alternatively, restart just the Admin Server component by typing the following command:

   $ZEUSHOME/admin/rc restart
Recent SOAP::Lite Versions

Versions of SOAP::Lite on or after 0.65_5 have a slightly different interface to earlier versions. When creating a SOAP::Lite connection, use the newer "ns" method instead of the previous "uri" method:

```perl
# 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');
```

Perl Deserializer Example

The SOAP::Lite module does not make use of the Traffic Manager’s WSDL specification and so does not know how to deserialize some enumerations used. For an example of this problem, see Chapter 3, “Using Perl SOAP::Lite.”

Tracing

To import the SOAP::Lite module with tracing enabled, use the following:

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

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

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

```perl
my $conn = SOAP::Lite;
$conn->readable(1);
```

While the messages sent by the client become more readable, this does not affect messages received from the server.

Debugging with C#

Fault Handlers

For details on how to inspect any SOAP Faults that are raised as a result of a server or transport error, see “Fault Handling” on page 15.

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 Traffic Manager Admin Server) unless the application is running from a trusted location. Remote filesystems locally mounted are considered untrusted, whereas local filesystems are trusted.

The location of your Control API application might affect whether it functions correctly or not.
Debugging with Java

Fault Handlers

For details on how to inspect any SOAP Faults that are raised as a result of a server or transport error, see “Fault Handling” on page 15.

Tracing

For full SOAP tracing, run your Control API application by typing the following command:

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

Alternatively, enable debugging within your application by adding the following line:

```
System.setProperty("javax.net.debug", "all");
```
CHAPTER 6  Function Reference

This chapter contains the full SOAP interface reference for the Traffic Manager Control API.

About the Control API Functions

Control API functions typically operate on lists of configurations. For example, the Virtual Server
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\(^1\).

For example, consider the method prototypes for the Virtual Server getRules() and addRules() functions:

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

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

getRules() takes a list of Virtual Server names and returns a list of VirtualServer.Rule arrays:

```java
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;
}
```

1.  The Traffic Manager WSDL interface specifications are located in $ZEUSHOME/zxtm/etc/wsdl/
The `VirtualServer.Rule` structure includes an enumerated type:

```java
enum VirtualServer.RuleRunFlag {
    # Run on every request or response
    run_every,
    # Run only on the first request or response
    only_first
}
```

Your SOAP toolkit represents 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. For details, see Chapter 3, “Using Perl SOAP::Lite.”
- 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.

For a worked example that illustrates the use of the methods, structures, and enumerations, see Chapter 4, “Sample Control API Applications.”

---

**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.

**Methods**

**addCompletionRules( names, rules ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add new rules to be run on the completion of a transaction 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.

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

**addCompletionRulesByLocation( location, names, rules ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add new rules to be run on the completion of a transaction 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. This is a location specific function, any action will operate on the specified location.

```java
void addCompletionRulesByLocation(
    String location,
    String[] names
    VirtualServer.Rule[][] rules
)
addCompressionMIMETypes( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

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

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

addCompressionMIMETypesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

For each named virtual server, add new MIME types to the list of types to compress. This is a location specific function, any action will operate on the specified location.

```java
void addCompressionMIMETypesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Add Space separated list of DNS zones

```java
void addDNSZones(
    String[] names
    String[][] values
)
```

addDNSZonesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add Space separated list of DNS zones. This is a location specific function, any action will operate on the specified location.

```java
void addDNSZonesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Add GLB Services used by the virtual server

```java
void addGLBServices(
    String[] names
    String[][] values
)
```
**addGLBServicesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add GLB Services used by the virtual server. This is a location specific function, any action will operate on the specified location.

```java
void addGLBServicesByLocation(
    String location
    String[] names
    String[][] values
)
```

**addHTTP2HeadersIndexBlacklist( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add a list of header names that should never be compressed using indexing.

```java
void addHTTP2HeadersIndexBlacklist(
    String[] names
    String[][] values
)
```

**addHTTP2HeadersIndexBlacklistByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add a list of header names that should never be compressed using indexing. This is a location specific function, any action will operate on the specified location.

```java
void addHTTP2HeadersIndexBlacklistByLocation(
    String location
    String[] names
    String[][] values
)
```

**addHTTP2HeadersIndexWhitelist( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add a list of header names that can be compressed using indexing when the default is to never index.

```java
void addHTTP2HeadersIndexWhitelist(
    String[] names
    String[][] values
)
```

**addHTTP2HeadersIndexWhitelistByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add a list of header names that can be compressed using indexing when the default is to never index. This is a location specific function, any action will operate on the specified location.

```java
void addHTTP2HeadersIndexWhitelistByLocation(
    String location
    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.

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

addResponseRulesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void addResponseRulesByLocation(
    String location
    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.

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

addRulesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void addRulesByLocation(
    String location
    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.

```java
void addSSLClientCertificateAuthorities(
    String[] names
    String[][] values
)`
**addSSLClientCertificateAuthoritiesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add new certificate authorities for validating client certificates for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void addSSLClientCertificateAuthoritiesByLocation(
    String location
    String[] names
    String[][] values
)
```

**addSSLNeverExpiringClientCertificateAuthorities( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add CAs for which any client certificate they validate is considered valid even if the client certificate's expiration date has passed, for each of the named virtual servers.

```java
void addSSLNeverExpiringClientCertificateAuthorities(
    String[] names
    String[][] values
)
```

**addSSLNeverExpiringClientCertificateAuthoritiesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add CAs for which any client certificate they validate is considered valid even if the client certificate's expiration date has passed, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void addSSLNeverExpiringClientCertificateAuthoritiesByLocation(
    String location
    String[] names
    String[][] values
)
```

**addSSLOCSPIssuers( names, ssl_ocsp_issuers ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Adds mappings between Certificate Authorities and OCSP responder settings. Certificates issued by these authorities will be verified with OCSP using these settings.

```java
void addSSLOCSPIssuers(
    String[] names
    VirtualServer.SSLOCSPIssuer[][] ssl_ocsp_issuers
)
```

**addSSLOCSPIssuersByLocation( location, names, ssl_ocsp_issuers ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Adds mappings between Certificate Authorities and OCSP responder settings. Certificates issued by these authorities will be verified with OCSP using these settings. This is a location specific function, any action will operate on the specified location.

```java
void addSSLOCSPIssuersByLocation(
    String location
    String[] names
    VirtualServer.SSLOCSPIssuer[][] ssl_ocsp_issuers
)
```
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.

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

addSSLSitesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void addSSLSitesByLocation(
    String location
    String[] names
    VirtualServer.SSLSite[][] ssl_sites
)
```

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

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

```java
void addSSLSitesEx(
    String[] names
    VirtualServer.SSLSiteAlt[][] ssl_sites
)
```

addSSLSitesExByLocation( location, names, ssl_sites ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Adds the specified SSLSiteAlt objects to the named virtual servers. These objects are mappings between destination addresses and the certificates used for SSL decryption those addresses. Each certificate is the name of an item in the SSL Certificates Catalog. This is a location specific function, any action will operate on the specified location.

```java
void addSSLSitesExByLocation(
    String location
    String[] names
    VirtualServer.SSLSiteAlt[][] ssl_sites
)
```

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

Add the set of HTTP header names for which corresponding values should be redacted from the metadata exported by this virtual server. from transaction export.

```java
void addTransactionExportHTTPHeaderBlacklist(
```
```java
String[] names
String[][] values
}

addTransactionExportHTTPHeaderBlacklistByLocation( location, names, values )
throws ObjectDoesNotExist, DeploymentError, InvalidInput

Add the set of HTTP header names for which corresponding values should be redacted from the metadata
exported by this virtual server, from transaction export. This is a location specific function, any action will
operate on the specified location.

void addTransactionExportHTTPHeaderBlacklistByLocation(
   String location
   String[] names
   String[][] values
}

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
}

deleteSSLSitesByLocation( location, 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. This is a
location specific function, any action will operate on the specified location.

void deleteSSLSitesByLocation(
   String location
   String[] names
   String[][] site_ips
```
VirtualServer Function Reference

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
}

editSSLSitesByLocation( location, 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. This is a
location specific function, any action will operate on the specified location.
void editSSLSitesByLocation{
    String location
    String[] names
    String[][] site_ips
    VirtualServer.SSLSite[][] ssl_sites
}

editSSLSitesEx( 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 certificates used for SSL
decryption those addresses. Each certificate is the name of an item in the SSL Certificates Catalog.
void editSSLSitesEx{
    String[] names
    String[][] site_ips
    VirtualServer.SSLSiteAlt[][] ssl_sites
}
editSSLSitesExByLocation( location, 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 certificates used for SSL decryption those addresses. Each certificate is the name of an item in the SSL Certificates Catalog. This is a location specific function, any action will operate on the specified location.

```java
void editSSLSitesExByLocation(
    String location
    String[] names
    String[][] site_ips
    VirtualServer.SSLSiteAlt[][] ssl_sites
)
```

getAddClusterClientIPHeader( names ) throws ObjectDoesNotExist

Get whether an '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.

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

getAddClusterClientIPHeaderByLocation( location, names ) throws ObjectDoesNotExist

Get whether an '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. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getAddClusterClientIPHeaderByLocation(
    String location
    String[] names
)
```

getAddXForwardedForHeader( names ) throws ObjectDoesNotExist

Get whether the remote client's IP address should be appended to the X-Forwarded-For header. The 'X-Forwarded-For' header contains the client's IP address.

```java
Boolean[] getAddXForwardedForHeader(
    String[] names
)
```

getAddXForwardedForHeaderByLocation( location, names ) throws ObjectDoesNotExist

Get whether the remote client's IP address should be appended to the X-Forwarded-For header. The 'X-Forwarded-For' header contains the client's IP address. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getAddXForwardedForHeaderByLocation(
    String location
    String[] names
)
VirtualServer Function Reference

getAddXForwardedProtoHeader(names) throws ObjectDoesNotExist

Get whether an 'X-Forwarded-Proto' header should be added to each HTTP request, for each of the named virtual servers. The 'X-Forwarded-Proto' header contains the protocol the client used to connect to the traffic manager.

```java
Boolean[] getAddXForwardedProtoHeader(
    String[] names
)
```

getAddXForwardedProtoHeaderByLocation(location, names) throws ObjectDoesNotExist

Get whether an 'X-Forwarded-Proto' header should be added to each HTTP request, for each of the named virtual servers. The 'X-Forwarded-Proto' header contains the protocol the client used to connect to the traffic manager. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getAddXForwardedProtoHeaderByLocation(
    String location,
    String[] names
)
```

getAptimizerEnabled(names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should optimize web content.

```java
Boolean[] getAptimizerEnabled(
    String[] names
)
```

getAptimizerEnabledByLocation(location, names) throws ObjectDoesNotExist

Get whether each of the named virtual servers should optimize web content. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getAptimizerEnabledByLocation(
    String location,
    String[] names
)
```

getAutodetectUpgradeHeaders(names) throws ObjectDoesNotExist

Get whether the traffic manager should check for HTTP responses that confirm an upgrade to the WebSockets protocol and automatically stop any protocol-specific processing for that connection when detected.

```java
Boolean[] getAutodetectUpgradeHeaders(
    String[] names
)
```

getAutodetectUpgradeHeadersByLocation(location, names) throws ObjectDoesNotExist

Get whether the traffic manager should check for HTTP responses that confirm an upgrade to the WebSockets protocol and automatically stop any protocol-specific processing for that connection when detected. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getAutodetectUpgradeHeadersByLocation(
    String location,
    String[] names
)
getBandwidthClass( names ) throws ObjectDoesNotExist

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

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

getBandwidthClassByLocation( location, names ) throws ObjectDoesNotExist

Get the Bandwidth Class that each of the named virtual servers uses. This is a location specific function, any action will operate on the specified location.

```java
String[] getBandwidthClassByLocation(
    String location
    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.

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

getBasicInfoByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.BasicInfo[] getBasicInfoByLocation(
    String location
    String[] names
)
```

getBypassDataPlaneAcceleration( names ) throws ObjectDoesNotExist

Get whether this service should, where possible, bypass data plane acceleration mechanisms.

```java
Boolean[] getBypassDataPlaneAcceleration(
    String[] names
)
```

getBypassDataPlaneAccelerationByLocation( location, names ) throws ObjectDoesNotExist

Get whether this service should, where possible, bypass data plane acceleration mechanisms. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getBypassDataPlaneAccelerationByLocation(
    String location
    String[] names
)
```
**getCloseWithRst( names ) throws ObjectDoesNotExist**

Get whether connections from clients should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state.

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

**getCloseWithRstByLocation( location, names ) throws ObjectDoesNotExist**

Get whether connections from clients should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getCloseWithRstByLocation(  
    String location  
    String[] names
)
```

**getCompletionRules( names ) throws ObjectDoesNotExist**

Get the rules that are run on the completion of a transaction for each of the named virtual servers.

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

**getCompletionRulesByLocation( location, names ) throws ObjectDoesNotExist**

Get the rules that are run on the completion of a transaction for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```
VirtualServer.Rule[][] getCompletionRulesByLocation(  
    String location  
    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
)
```

**getCompressUnknownSizeByLocation( location, names ) throws ObjectDoesNotExist**

Get whether each of the named virtual servers should compress documents with no given size. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getCompressUnknownSizeByLocation(  
    String location  
    String[] names
)
```

**getCompressionETagRewrite( names ) throws ObjectDoesNotExist**

Get how the ETag header should be manipulated when compressing content.
VirtualServer.CompressionETagRewrite[] getCompressionETagRewrite(
    String[] names
)

getCompressionETagRewriteByLocation( location, names ) throws ObjectDoesNotExist

Get how the ETag header should be manipulated when compressing content. This is a location specific function, any action will operate on the specified location.

VirtualServer.CompressionETagRewrite[] getCompressionETagRewriteByLocation(
    String location
    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
)

getCompressionEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should compress web pages before sending to the client. This is a location specific function, any action will operate on the specified location.

Boolean[] getCompressionEnabledByLocation(
    String location
    String[] names
)

getCompressionLevel( names ) throws ObjectDoesNotExist

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

Unsigned Integer[] getCompressionLevel(
    String[] names
)

getCompressionLevelByLocation( location, names ) throws ObjectDoesNotExist

Get the gzip compression level, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getCompressionLevelByLocation(
    String location
    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
)
**getCompressionMIMETypesByLocation( location, names ) throws ObjectDoesNotExist**

Get the list of MIME types to compress, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
String[][] getCompressionMIMETypesByLocation(
    String location
    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'.

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

**getCompressionMaxSizeByLocation( location, 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'. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getCompressionMaxSizeByLocation(
    String location
    String[] names
)
```

**getCompressionMinSize( names ) throws ObjectDoesNotExist**

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

```java
Unsigned Integer[] getCompressionMinSize(
    String[] names
)
```

**getCompressionMinSizeByLocation( location, names ) throws ObjectDoesNotExist**

Get the minimum document size to compress, in bytes, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getCompressionMinSizeByLocation(
    String location
    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.

```java
Unsigned Integer[] getConnectTimeout(
    String[] names
)
```
getConnectTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getConnectTimeoutByLocation(
    String location
    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
)

getCookieDomainRewriteModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

VirtualServer.CookieDomainRewriteMode[] getCookieDomainRewriteModeByLocation(
    String location
    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
)

getCookieNamedDomainByLocation( location, names ) throws ObjectDoesNotExist

Get the domain to use when rewriting cookie domains, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

String[] getCookieNamedDomainByLocation(
    String location
    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
)
getCookiePathRewriteByLocation( location, names ) throws ObjectDoesNotExist

For each of the named virtual servers, get the regex and replacement for rewriting the path portion of a cookie. This is a location specific function, any action will operate on the specified location.

VirtualServer.RegexReplacement[] getCookiePathRewriteByLocation(
    String location
    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
)

getCookieSecureFlagRewriteModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

VirtualServer.CookieSecureFlagRewriteMode[] getCookieSecureFlagRewriteModeByLocation(
    String location
    String[] names
)

getDNSEdnsClientSubnet( names ) throws ObjectDoesNotExist

Get whether use of EDNS client subnet option is enabled

Boolean[] getDNSEdnsClientSubnet(
    String[] names
)

getDNSEdnsClientSubnetByLocation( location, names ) throws ObjectDoesNotExist

Get whether use of EDNS client subnet option is enabled. This is a location specific function, any action will operate on the specified location.

Boolean[] getDNSEdnsClientSubnetByLocation(
    String location
    String[] names
)

getDNSEdnsUdpsize( names ) throws ObjectDoesNotExist

Get EDNS UDP size advertised in responses

Unsigned Integer[] getDNSEdnsUdpsize(
    String[] names
)
**getDNSEdnsUdpsizeByLocation( location, names ) throws ObjectDoesNotExist**

Get EDNS UDP size advertised in responses This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getDNSEdnsUdpsizeByLocation(
    String location
    String[] names
)
```

**getDNSMaxUdpsize( names ) throws ObjectDoesNotExist**

Get Maximum UDP answer size

```java
Unsigned Integer[] getDNSMaxUdpsize(
    String[] names
)
```

**getDNSMaxUdpsizeByLocation( location, names ) throws ObjectDoesNotExist**

Get Maximum UDP answer size This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getDNSMaxUdpsizeByLocation(
    String location
    String[] names
)
```

**getDNSRecordsetOrder( names ) throws ObjectDoesNotExist**

Get order of records in a DNS response

```java
VirtualServer.DNSRecordsetOrder[] getDNSRecordsetOrder(
    String[] names
)
```

**getDNSRecordsetOrderByLocation( location, names ) throws ObjectDoesNotExist**

Get order of records in a DNS response This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.DNSRecordsetOrder[] getDNSRecordsetOrderByLocation(
    String location
    String[] names
)
```

**getDNSVerbose( names ) throws ObjectDoesNotExist**

Get whether built-in DNS server should log verbose messages

```java
Boolean[] getDNSVerbose(
    String[] names
)
```

**getDNSVerboseByLocation( location, names ) throws ObjectDoesNotExist**

Get whether built-in DNS server should log verbose messages This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getDNSVerboseByLocation(
    String location
    String[] names
)
```
getDNSZones( names ) throws ObjectDoesNotExist
Get Space separated list of DNS zones
String[] names
)

getDNSZonesByLocation( location, names ) throws ObjectDoesNotExist
Get Space separated list of DNS zones This is a location specific function, any action will operate on the specified location.
String[][] names
)

setDefaultPool( names ) throws ObjectDoesNotExist
Get the default Pool that traffic is sent to for each of the named virtual servers.
String[] defaultPool

setDefaultPoolByLocation( location, names ) throws ObjectDoesNotExist
Get the default Pool that traffic is sent to for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.
String[] defaultPoolByLocation

getEnabled( names ) throws ObjectDoesNotExist
Get whether each of the named virtual servers is enabled (i.e. serving traffic).
Boolean[] enabled

getEnabledByLocation( location, names ) throws ObjectDoesNotExist
Get whether each of the named virtual servers is enabled (i.e. serving traffic). This is a location specific function, any action will operate on the specified location.
Boolean[] enabledByLocation

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.

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

ger得不到FileByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
String[] getErrorFileByLocation(
    String location
    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.

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

getFTPDataSourcePortByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getFTPDataSourcePortByLocation(
    String location
    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.

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

getFTPForceClientSecureByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getFTPForceClientSecureByLocation(
    String location
    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
)

getFTPForceServerSecureByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Boolean[] getFTPForceServerSecureByLocation(
    String location
    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
)

getFTPPortRangeByLocation( location, names ) throws ObjectDoesNotExist

Get the port range used for FTP data connections for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

VirtualServer.FTPPortRange[] getFTPPortRangeByLocation(
    String location
    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
)

getFTPSSLDataByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should use SSL on the data connection as well as the control connection. This is a location specific function, any action will operate on the specified location.

Boolean[] getFTPSSLDataByLocation(
    String location
    String[] names
)
getGLBServices( names ) throws ObjectDoesNotExist

Get GLB Services used by the virtual server

```java
String[][] getGLBServices(
    String[] names
)
```

getGLBServicesByLocation( location, names ) throws ObjectDoesNotExist

Get GLB Services used by the virtual server This is a location specific function, any action will operate on the specified location.

```java
String[][] getGLBServicesByLocation(
    String location
    String[] names
)
```

getHTTP2ConnectTimeout( names ) throws ObjectDoesNotExist

Get the time to wait for a request on a new HTTP/2 connection, in seconds. If no request is received in this time, the connection will be closed. This setting overrides 'connect_timeout', and uses the value of 'connect_timeout' if set to zero.

```java
Unsigned Integer[] getHTTP2ConnectTimeout(
    String[] names
)
```

getHTTP2ConnectTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get the time to wait for a request on a new HTTP/2 connection, in seconds. If no request is received in this time, the connection will be closed. This setting overrides 'connect_timeout', and uses the value of 'connect_timeout' if set to zero. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getHTTP2ConnectTimeoutByLocation(
    String location
    String[] names
)
```

getHTTP2DataFrameSize( names ) throws ObjectDoesNotExist

Get the preferred HTTP/2 data frame size

```java
Unsigned Integer[] getHTTP2DataFrameSize(
    String[] names
)
```

getHTTP2DataFrameSizeByLocation( location, names ) throws ObjectDoesNotExist

Get the preferred HTTP/2 data frame size This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getHTTP2DataFrameSizeByLocation(
    String location
    String[] names
)
```
**getHTTP2Enabled( names ) throws ObjectDoesNotExist**

Get allows the HTTP/2 protocol to be used.

```java
Boolean[] getHTTP2Enabled(
    String[] names
)
```

**getHTTP2EnabledByLocation( location, names ) throws ObjectDoesNotExist**

Get allows the HTTP/2 protocol to be used. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getHTTP2EnabledByLocation(
    String location
    String[] names
)
```

**getHTTP2HeaderTableSize( names ) throws ObjectDoesNotExist**

Get the amount of memory allowed for header compression

```java
Unsigned Integer[] getHTTP2HeaderTableSize(
    String[] names
)
```

**getHTTP2HeaderTableSizeByLocation( location, names ) throws ObjectDoesNotExist**

Get the amount of memory allowed for header compression. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getHTTP2HeaderTableSizeByLocation(
    String location
    String[] names
)
```

**getHTTP2HeadersIndexBlacklist( names ) throws ObjectDoesNotExist**

Get a list of header names that should never be compressed using indexing.

```java
String[][] getHTTP2HeadersIndexBlacklist(
    String[] names
)
```

**getHTTP2HeadersIndexBlacklistByLocation( location, names ) throws ObjectDoesNotExist**

Get a list of header names that should never be compressed using indexing. This is a location specific function, any action will operate on the specified location.

```java
String[][] getHTTP2HeadersIndexBlacklistByLocation(
    String location
    String[] names
)
```
getHTTP2HeadersIndexDefault( names ) throws ObjectDoesNotExist

Get whether all HTTP/2 headers should be compressed using indexing, except those specified in the blacklist, or whether all HTTP/2 headers should not be compressed using indexing, except those specified in the whitelist.

```java
Boolean[] getHTTP2HeadersIndexDefault(
    String[] names
)
```

getHTTP2HeadersIndexDefaultByLocation( location, names ) throws ObjectDoesNotExist

Get whether all HTTP/2 headers should be compressed using indexing, except those specified in the blacklist, or whether all HTTP/2 headers should not be compressed using indexing, except those specified in the whitelist. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getHTTP2HeadersIndexDefaultByLocation(
    String location,
    String[] names
)
```

getHTTP2HeadersIndexWhitelist( names ) throws ObjectDoesNotExist

Get a list of header names that can be compressed using indexing when the default is to never index.

```java
String[][] getHTTP2HeadersIndexWhitelist(
    String[] names
)
```

getHTTP2HeadersIndexWhitelistByLocation( location, names ) throws ObjectDoesNotExist

Get a list of header names that can be compressed using indexing when the default is to never index. This is a location specific function, any action will operate on the specified location.

```java
String[][] getHTTP2HeadersIndexWhitelistByLocation(
    String location,
    String[] names
)
```

getHTTP2HeadersSizeLimit( names ) throws ObjectDoesNotExist

Get the maximum size, in bytes, of decompressed headers for an HTTP/2 request.

```java
Unsigned Integer[] getHTTP2HeadersSizeLimit(
    String[] names
)
```

getHTTP2HeadersSizeLimitByLocation( location, names ) throws ObjectDoesNotExist

Get the maximum size, in bytes, of decompressed headers for an HTTP/2 request. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getHTTP2HeadersSizeLimitByLocation(
    String location,
    String[] names
)
```
getHTTP2IdleTimeoutNoStreams( names ) throws ObjectDoesNotExist

Get the time to wait for an HTTP/2 request on a previously used HTTP/2 connection with no open streams.

Unsigned Integer[] getHTTP2IdleTimeoutNoStreams(
    String[] names
)

getHTTP2IdleTimeoutNoStreamsByLocation( location, names ) throws ObjectDoesNotExist

Get the time to wait for an HTTP/2 request on a previously used HTTP/2 connection with no open streams. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getHTTP2IdleTimeoutNoStreamsByLocation(
    String location
    String[] names
)

getHTTP2IdleTimeoutOpenStreams( names ) throws ObjectDoesNotExist

Get the time to wait for data on an HTTP/2 connection with open streams that haven't sent data recently

Unsigned Integer[] getHTTP2IdleTimeoutOpenStreams(
    String[] names
)

getHTTP2IdleTimeoutOpenStreamsByLocation( location, names ) throws ObjectDoesNotExist

Get the time to wait for data on an HTTP/2 connection with open streams that haven't sent data recently. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getHTTP2IdleTimeoutOpenStreamsByLocation(
    String location
    String[] names
)

getHTTP2MaxConcurrentStreams( names ) throws ObjectDoesNotExist

Get the number of concurrent streams allowed

Unsigned Integer[] getHTTP2MaxConcurrentStreams(
    String[] names
)

getHTTP2MaxConcurrentStreamsByLocation( location, names ) throws ObjectDoesNotExist

Get the number of concurrent streams allowed. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getHTTP2MaxConcurrentStreamsByLocation(
    String location
    String[] names
)

getHTTP2MaxFrameSize( names ) throws ObjectDoesNotExist

Get the maximum HTTP/2 frame size
Unsigned Integer[] getHTTP2MaxFrameSize(
    String[] names
)

getHTTP2MaxFrameSizeByLocation( location, names ) throws ObjectDoesNotExist
Get the maximum HTTP/2 frame size This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getHTTP2MaxFrameSizeByLocation(
    String location
    String[] names
)

getHTTP2MaxHeaderPadding( names ) throws ObjectDoesNotExist
Get the maximum size, in bytes, of random-length padding to add to HTTP/2 header frames
Unsigned Integer[] getHTTP2MaxHeaderPadding(
    String[] names
)

getHTTP2MaxHeaderPaddingByLocation( location, names ) throws ObjectDoesNotExist
Get the maximum size, in bytes, of random-length padding to add to HTTP/2 header frames This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getHTTP2MaxHeaderPaddingByLocation(
    String location
    String[] names
)

getHTTP2MergeCookieHeaders( names ) throws ObjectDoesNotExist
Get whether Cookie headers received from an HTTP/2 client should be merged into a single Cookie header before forwarding to an HTTP/1.1 server.
Boolean[] getHTTP2MergeCookieHeaders(
    String[] names
)

getHTTP2MergeCookieHeadersByLocation( location, names ) throws ObjectDoesNotExist
Get whether Cookie headers received from an HTTP/2 client should be merged into a single Cookie header before forwarding to an HTTP/1.1 server. This is a location specific function, any action will operate on the specified location.
Boolean[] getHTTP2MergeCookieHeadersByLocation(
    String location
    String[] names
)

getHTTP2StreamWindowSize( names ) throws ObjectDoesNotExist
Get the flow control window size
Unsigned Integer[] getHTTP2StreamWindowSize(
    String[] names
)
getHTTP2StreamWindowSizeByLocation( location, names ) throws ObjectDoesNotExist

Get the flow control window size. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getHTTP2StreamWindowSizeByLocation(
    String location
    String[] names
)

getHttpChunkOverheadForwarding( names ) throws ObjectDoesNotExist

Get how to handle forwarding of data that is pure HTTP chunking overhead.

VirtualServer.HttpChunkOverheadForwarding[] getHttpChunkOverheadForwarding(
    String[] names
)

getHttpChunkOverheadForwardingByLocation( location, names ) throws ObjectDoesNotExist

Get how to handle forwarding of data that is pure HTTP chunking overhead. This is a location specific function, any action will operate on the specified location.

VirtualServer.HttpChunkOverheadForwarding[] getHttpChunkOverheadForwardingByLocation(
    String location
    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
)

getKeepaliveByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should allow clients to maintain keepalive connections. This is a location specific function, any action will operate on the specified location.

Boolean[] getKeepaliveByLocation(
    String location
    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
)
**getKeepaliveTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getKeepaliveTimeoutByLocation(
    String location
    String[] names
)
```

**getKerberosProtocolTransitionEnabled( names ) throws ObjectDoesNotExist**

Get whether each of the named virtual servers should use Kerberos Protocol Transition.

```java
Boolean[] getKerberosProtocolTransitionEnabled(
    String[] names
)
```

**getKerberosProtocolTransitionEnabledByLocation( location, names ) throws ObjectDoesNotExist**

Get whether each of the named virtual servers should use Kerberos Protocol Transition. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getKerberosProtocolTransitionEnabledByLocation(
    String location
    String[] names
)
```

**getKerberosProtocolTransitionPrincipal( names ) throws ObjectDoesNotExist**

Get the Kerberos principal that each of the named virtual servers uses to perform Kerberos Protocol Transition.

```java
String[] getKerberosProtocolTransitionPrincipal(
    String[] names
)
```

**getKerberosProtocolTransitionPrincipalByLocation( location, names ) throws ObjectDoesNotExist**

Get the Kerberos principal that each of the named virtual servers uses to perform Kerberos Protocol Transition. This is a location specific function, any action will operate on the specified location.

```java
String[] getKerberosProtocolTransitionPrincipalByLocation(
    String location
    String[] names
)
```

**getKerberosProtocolTransitionTarget( names ) throws ObjectDoesNotExist**

Get the Kerberos principal name of the service that each of the named virtual servers targets.

```java
String[] getKerberosProtocolTransitionTarget(
    String[] names
)
```
VirtualServer

**getKerberosProtocolTransitionTargetByLocation**

Get the Kerberos principal name of the service that each of the named virtual servers targets. This is a location specific function, any action will operate on the specified location.

```java
String[] getKerberosProtocolTransitionTargetByLocation(
    String location
    String[] names
)
```

**getL4AccelRSTOnServiceFailure**

Get whether the virtual server should send a TCP RST packet or ICMP error message if a service is unavailable, or if an established connection to a node fails.

```java
Boolean[] getL4AccelRSTOnServiceFailure(
    String[] names
)
```

**getL4AccelRSTOnServiceFailureByLocation**

Get whether the virtual server should send a TCP RST packet or ICMP error message if a service is unavailable, or if an established connection to a node fails. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getL4AccelRSTOnServiceFailureByLocation(
    String location
    String[] names
)
```

**getL4AccelServiceIPSNAT**

Get whether or not backend connections should be configured to use the ingress service IP as the source IP for the back-end connection when Source NAT is enabled for the pool used by the service.

```java
Boolean[] getL4AccelServiceIPSNAT(
    String[] names
)
```

**getL4AccelServiceIPSNATByLocation**

Get whether or not backend connections should be configured to use the ingress service IP as the source IP for the back-end connection when Source NAT is enabled for the pool used by the service. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getL4AccelServiceIPSNATByLocation(
    String location
    String[] names
)
```

**getL4AccelStateSync**

Get whether the state of active connections will be synchronized across the cluster for L4Accel services, such that connections will persist in the event of a failover.

```java
Boolean[] getL4AccelStateSync(
    String[] names
)
```
getL4AccelStateSyncByLocation( location, names ) throws ObjectDoesNotExist
Get whether the state of active connections will be synchronized across the cluster for L4Accel services, such that connections will persist in the event of a failover. This is a location specific function, any action will operate on the specified location.

Boolean[] getL4AccelStateSyncByLocation(
    String location
    String[] names
)

getL4AccelTCPMaxSegmentLifetime( names ) throws ObjectDoesNotExist
Get the maximum segment lifetime, in seconds, of a TCP segment being handled by the traffic manager. This setting determines for how long information about a connection will be retained after receiving a two-way FIN or RST.

Unsigned Integer[] getL4AccelTCPMaxSegmentLifetime(
    String[] names
)

getL4AccelTCPMaxSegmentLifetimeByLocation( location, names ) throws ObjectDoesNotExist
Get the maximum segment lifetime, in seconds, of a TCP segment being handled by the traffic manager. This setting determines for how long information about a connection will be retained after receiving a two-way FIN or RST. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getL4AccelTCPMaxSegmentLifetimeByLocation(
    String location
    String[] names
)

getL4AccelTimeout( names ) throws ObjectDoesNotExist
Get the number of seconds after which a connection will be closed if no further packets have been received on it.

Unsigned Integer[] getL4AccelTimeout(
    String[] names
)

getL4AccelTimeoutByLocation( location, names ) throws ObjectDoesNotExist
Get the number of seconds after which a connection will be closed if no further packets have been received on it. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getL4AccelTimeoutByLocation(
    String location
    String[] names
)
**getL4AccelUDPCountRequests(names) throws ObjectDoesNotExist**

Get whether a connection should be closed when the number of UDP response datagrams received from the server is equal to the number of request datagrams that have been sent by the client. If set to No the connection will be closed after the first response has been received from the server. This setting takes precedence over l4accel!optimized_aging setting.

```java
Boolean[] getL4AccelUDPCountRequests(
    String[] names
)
```

**getL4AccelUDPCountRequestsByLocation(location, names) throws ObjectDoesNotExist**

Get whether a connection should be closed when the number of UDP response datagrams received from the server is equal to the number of request datagrams that have been sent by the client. If set to No the connection will be closed after the first response has been received from the server. This setting takes precedence over l4accel!optimized_aging setting. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getL4AccelUDPCountRequestsByLocation(
    String location
    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.

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

**getListenAddressesByLocation(location, 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. This is a location specific function, any action will operate on the specified location.

```java
String[][] getListenAddressesByLocation(
    String location
    String[] names
)
```

**getListenOnAllAddresses(names) throws ObjectDoesNotExist**

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

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

**getListenOnAllAddressesByLocation(location, names) throws ObjectDoesNotExist**

For each of the named virtual servers, gets whether the virtual server is listening on all IP addresses This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getListenOnAllAddressesByLocation(
)```
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.

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

getListenTrafficIPGroupsByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
String[][] getListenTrafficIPGroupsByLocation(
    String location
    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.

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

getLocationDefaultRewriteModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.LocationDefaultRewriteMode[] getLocationDefaultRewriteModeByLocation(
    String location
    String[] names
)
```

getLocationRewrite( names ) throws ObjectDoesNotExist
For each of the named virtual servers, get the regex, and replacement for rewriting any 'Location' headers.

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

getLocationRewriteByLocation( location, names ) throws ObjectDoesNotExist
For each of the named virtual servers, get the regex, and replacement for rewriting any 'Location' headers. This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.RegexReplacement[] getLocationRewriteByLocation(
    String location
    String[] names
)
```
VirtualServer

String location
String[] names

getLogClientConnectionFailures( names ) throws ObjectDoesNotExist

Get whether the virtual server will log client connection failures.

Boolean[] getLogClientConnectionFailures(
    String[] names
)

getLogClientConnectionFailuresByLocation( location, names ) throws ObjectDoesNotExist

Get whether the virtual server will log client connection failures. This is a location specific function, any action will operate on the specified location.

Boolean[] getLogClientConnectionFailuresByLocation(
    String location
    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
)

getLogEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should log each connection to a disk on the file system. This is a location specific function, any action will operate on the specified location.

Boolean[] getLogEnabledByLocation(
    String location
    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
)

getLogFilenameByLocation( location, names ) throws ObjectDoesNotExist

Get the name of the file used to store request logs, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

String[] getLogFilenameByLocation(
    String location
    String[] names
)
**getLogFormat( names ) throws ObjectDoesNotExist**

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

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

**getLogFormatByLocation( location, names ) throws ObjectDoesNotExist**

Get the log file format for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
String[] getLogFormatByLocation(
    String location
    String[] names
)
```

**getLogSSLFailures( names ) throws ObjectDoesNotExist**

Get whether the virtual server will log ssl failures.

```java
Boolean[] getLogSSLFailures(
    String[] names
)
```

**getLogSSLFailuresByLocation( location, names ) throws ObjectDoesNotExist**

Get whether the virtual server will log ssl failures. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getLogSSLFailuresByLocation(
    String location
    String[] names
)
```

**getLogSaveAll( names ) throws ObjectDoesNotExist**

Get whether to log all connections by default, or log no connections by default.

```java
Boolean[] getLogSaveAll(
    String[] names
)
```

**getLogSaveAllByLocation( location, names ) throws ObjectDoesNotExist**

Get whether to log all connections by default, or log no connections by default. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getLogSaveAllByLocation(
    String location
    String[] names
)
```

**getLogServerConnectionFailures( names ) throws ObjectDoesNotExist**

Get whether the virtual server will log server connection failures.

```java
Boolean[] getLogServerConnectionFailures(
    String[] names
)
```
**getLogServerConnectionFailuresByLocation( location, names ) throws ObjectDoesNotExist**

Get whether the virtual server will log server connection failures. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getLogServerConnectionFailuresByLocation(
    String location
    String[] names
)
```

**getLogSessionPersistenceVerbose( names ) throws ObjectDoesNotExist**

Get whether the virtual server will log session persistence events.

```java
Boolean[] getLogSessionPersistenceVerbose(
    String[] names
)
```

**getLogSessionPersistenceVerboseByLocation( location, names ) throws ObjectDoesNotExist**

Get whether the virtual server will log session persistence events. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getLogSessionPersistenceVerboseByLocation(
    String location
    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.

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

**getMIMEAutoDetectByLocation( location, names ) throws ObjectDoesNotExist**

Get whether each of the named virtual servers should auto-detect MIME types if the server does not provide them. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getMIMEAutoDetectByLocation(
    String location
    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.

```java
String[] getMIMEDefaultType(
    String[] names
)
```
getMIMEDefaultTypeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

String[] getMIMEDefaultTypeByLocation{
    String location
    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
}

getMaxClientBufferByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxClientBufferByLocation{
    String location
    String[] names
}

getMaxConcurrentConnections( names ) throws ObjectDoesNotExist

Get sets the maximum number of concurrent TCP connections this virtual server will accept. A value of 0 will allow unlimited concurrent TCP connections to this virtual server.

Unsigned Integer[] getMaxConcurrentConnections{
    String[] names
}

getMaxConcurrentConnectionsByLocation( location, names ) throws ObjectDoesNotExist

Get sets the maximum number of concurrent TCP connections this virtual server will accept. A value of 0 will allow unlimited concurrent TCP connections to this virtual server. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxConcurrentConnectionsByLocation{
    String location
    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
}
VirtualServer Function Reference

**getMaxServerBufferByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getMaxServerBufferByLocation(
    String location
    String[] names
)
```

**getMaxTransactionDuration( names ) throws ObjectDoesNotExist**

Get the total amount of time a transaction can take, zero means forever.

```java
Unsigned Integer[] getMaxTransactionDuration(
    String[] names
)
```

**getMaxTransactionDurationByLocation( location, names ) throws ObjectDoesNotExist**

Get the total amount of time a transaction can take, zero means forever. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getMaxTransactionDurationByLocation(
    String location
    String[] names
)
```

**getNote( names ) throws ObjectDoesNotExist**

Get the note for each of the named virtual servers.

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

**getPort( names ) throws ObjectDoesNotExist**

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

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

**getPortByLocation( location, names ) throws ObjectDoesNotExist**

Get the port that each of the named virtual servers listens on for incoming connections. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getPortByLocation(
    String location
    String[] names
)
```

**getProtection( names ) throws ObjectDoesNotExist**

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

```java
String[] getProtection(
    String[] names
)
getProtectionByLocation( location, names ) throws ObjectDoesNotExist
Get the Service Protection Settings that are used to protect each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```
String[] getProtectionByLocation(
    String location
    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 the traffic manager is responding to the request itself, enabling this setting will cause the traffic manager to continue writing the response even after it has received a FIN from the client.

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

getProxyCloseByLocation( location, 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 the traffic manager is responding to the request itself, enabling this setting will cause the traffic manager to continue writing the response even after it has received a FIN from the client. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getProxyCloseByLocation(
    String location
    String[] names
)
```

getProxyProtocol( names ) throws ObjectDoesNotExist
Get expect connections to the traffic manager to be prefixed with a PROXY protocol header.

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

getProxyProtocolByLocation( location, names ) throws ObjectDoesNotExist
Get expect connections to the traffic manager to be prefixed with a PROXY protocol header. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getProxyProtocolByLocation(
    String location
    String[] names
)
VirtualServer

}  

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
)

getRTSPPortRangeByLocation( location, names ) throws ObjectDoesNotExist

Get the port range used for RTSP streaming data connections, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

VirtualServer.PortRange[] getRTSPPortRangeByLocation(
    String location
    String[] names
)

getRTSPStreamingTimeout( names ) throws ObjectDoesNotExist

Get the time, in seconds, after which data-streams associated with RTSP connections timeout if no data is transmitted.

Unsigned Integer[] getRTSPStreamingTimeout(
    String[] names
)

getRTSPStreamingTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get the time, in seconds, after which data-streams associated with RTSP connections timeout if no data is transmitted. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getRTSPStreamingTimeoutByLocation(
    String location
    String[] names
)

getRecentConnsEnabled( names ) throws ObjectDoesNotExist

Get whether or not any connections handled by this virtual server should be shown on the Connections page.

Boolean[] getRecentConnsEnabled(
    String[] names
)

getRecentConnsEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether or not any connections handled by this virtual server should be shown on the Connections page. This is a location specific function, any action will operate on the specified location.

Boolean[] getRecentConnsEnabledByLocation(
    String location
    String[] names
)
getRecentConnsSaveAll( names ) throws ObjectDoesNotExist
Get whether or not all connections handled by this virtual server should be shown on the Connections page.

Boolean[] getRecentConnsSaveAll(
    String[] names
)

getRecentConnsSaveAllByLocation( location, names ) throws ObjectDoesNotExist
Get whether or not all connections handled by this virtual server should be shown on the Connections page. This is a location specific function, any action will operate on the specified location.

Boolean[] getRecentConnsSaveAllByLocation(
    String location
    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
)

getRequestSyslogEnabledByLocation( location, names ) throws ObjectDoesNotExist
Get whether each of the named virtual servers should log each connection to a remote syslog server. This is a location specific function, any action will operate on the specified location.

Boolean[] getRequestSyslogEnabledByLocation(
    String location
    String[] names
)

getRequestSyslogFormat( names ) throws ObjectDoesNotExist
Get the remote log line format for each of the named virtual servers.

String[] getRequestSyslogFormat(
    String[] names
)

getRequestSyslogFormatByLocation( location, names ) throws ObjectDoesNotExist
Get the remote log line format for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

String[] getRequestSyslogFormatByLocation(
    String location
    String[] names
)

getRequestSyslogIPEndpoint( names ) throws ObjectDoesNotExist
Get the remote syslog endpoint for each of the named virtual servers.

String[] getRequestSyslogIPEndpoint(
    String[] names
)
getRequestSyslogIPEndpointByLocation( location, names ) throws ObjectDoesNotExist

Get the remote syslog endpoint for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
String[] getRequestSyslogIPEndpointByLocation(
    String location
    String[] names
)
```

ggetRequestSyslogMessageLenLimit( names ) throws ObjectDoesNotExist

Get syslog message length limit.

```java
Unsigned Integer[] getRequestSyslogMessageLenLimit(
    String[] names
)
```

ggetRequestSyslogMessageLenLimitByLocation( location, names ) throws ObjectDoesNotExist

Get syslog message length limit. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getRequestSyslogMessageLenLimitByLocation(
    String location
    String[] names
)
```

ggetRequestTracingEnabled( names ) throws ObjectDoesNotExist

Get whether to record a detailed list of processing history for each request.

```java
Boolean[] getRequestTracingEnabled(
    String[] names
)
```

ggetRequestTracingEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether to record a detailed list of processing history for each request. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getRequestTracingEnabledByLocation(
    String location
    String[] names
)
```

ggetRequestTracingIO( names ) throws ObjectDoesNotExist

Get whether to record a detailed list of every IO event in the processing history for each request.

```java
Boolean[] getRequestTracingIO(
    String[] names
)
```
**getRequestTracingIOByLocation( location, names ) throws ObjectDoesNotExist**

Get whether to record a detailed list of every IO event in the processing history for each request. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getRequestTracingIOByLocation(
    String location
    String[] names
)
```

**getResponseRules( names ) throws ObjectDoesNotExist**

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

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

**getResponseRulesByLocation( location, names ) throws ObjectDoesNotExist**

Get the rules that are run on server responses for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.Rule[][] getResponseRulesByLocation(
    String location
    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.

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

**getRewriteSIPURIByLocation( location, names ) throws ObjectDoesNotExist**

Get whether the Request-URI of SIP requests will be replaced with the selected back-end node's address. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getRewriteSIPURIByLocation(
    String location
    String[] names
)
```

**getRules( names ) throws ObjectDoesNotExist**

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

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

**getRulesByLocation( location, names ) throws ObjectDoesNotExist**

Get the rules that are run on client requests for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.Rule[][] getRulesByLocation(
    String location
```
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
)
```

getSIPDangerousRequestModeByLocation( location, names ) throws ObjectDoesNotExist

Get what should be done with requests that contain body data and should be routed to an external IP. This is a location specific function, any action will operate on the specified location.

```
VirtualServer.SIPDangerousRequestMode[] getSIPDangerousRequestModeByLocation(
    String location
    String[] names
)
```

g getSIPFollowRoute( names ) throws ObjectDoesNotExist

Get whether to follow routing information in SIP requests.

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

g getSIPFollowRouteByLocation( location, names ) throws ObjectDoesNotExist

Get whether to follow routing information in SIP requests. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getSIPFollowRouteByLocation(
    String location
    String[] names
)
```

getSIPMaxConnectionMemory( names ) throws ObjectDoesNotExist

Get maximum memory per connection.

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

getSIPMaxConnectionMemoryByLocation( location, names ) throws ObjectDoesNotExist

Get maximum memory per connection. This is a location specific function, any action will operate on the specified location.

```
Unsigned Integer[] getSIPMaxConnectionMemoryByLocation(
    String location
    String[] names
)
```
getSIPMode( names ) throws ObjectDoesNotExist

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

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

g getSIPModeByLocation( location, names ) throws ObjectDoesNotExist

Get which mode of operation the SIP virtual server should run in. This is a location specific function, any action will operate on the specified location.

VirtualServer.SIPMode[] getSIPModeByLocation(
    String location
    String[] names
)

g 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
)

g getSIPPortRangeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

VirtualServer.PortRange[] getSIPPortRangeByLocation(
    String location
    String[] names
)

g 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
)

g getSIPStreamingTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get the time, in seconds, after which a UDP stream will timeout if it has not seen any data. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getSIPStreamingTimeoutByLocation(
    String location
    String[] names
)

g getSIPTimeoutMessages( names ) throws ObjectDoesNotExist

Get send a timed out response to the client and CANCEL request to the server when a transaction times out.
Boolean[] getSIPTimeoutMessages(
    String[] names
)

getSIPTimeoutMessagesByLocation( location, names ) throws ObjectDoesNotExist

Get send a timed out response to the client and CANCEL request to the server when a transaction times out. This is a location specific function, any action will operate on the specified location.

Boolean[] getSIPTimeoutMessagesByLocation(
    String location
    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
)

getSSLCertificateByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

String[] getSSLCertificateByLocation(
    String location
    String[] names
)

getSSLCertificates( names ) throws ObjectDoesNotExist

Get the name of the default SSL Certificate, as well as alt certificates, that can be used for SSL decryption for each of the named virtual servers. This is the name of an item in the SSL Certificates Catalog.

String[][] getSSLCertificates(
    String[] names
)

getSSLCertificatesByLocation( location, names ) throws ObjectDoesNotExist

Get the name of the default SSL Certificate, as well as alt certificates, that can be used for SSL decryption for each of the named virtual servers. This is the name of an item in the SSL Certificates Catalog. This is a location specific function, any action will operate on the specified location.

String[][] getSSLCertificatesByLocation(
    String location
    String[] names
)

getSSLCiphers( names ) throws ObjectDoesNotExist

Get the ciphers allowed for connections to this virtual server

String[] getSSLCiphers(}
Function Reference

VirtualServer

String[] names

getSSLCiphersByLocation( location, names ) throws ObjectDoesNotExist
Get the ciphers allowed for connections to this virtual server. This is a location specific function, any action will operate on the specified location.

String[] getSSLCiphersByLocation(
    String location
    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
)

getSSLClientCertificateAuthoritiesByLocation( location, names ) throws ObjectDoesNotExist
Get the certificate authorities that are trusted for validating client certificates, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

String[][] getSSLClientCertificateAuthoritiesByLocation(
    String location
    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
)

getSSLClientCertificateHeadersByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLClientCertificateHeaders[] getSSLClientCertificateHeadersByLocation(
    String location
    String[] names
)

getSSLDecrypt( names ) throws ObjectDoesNotExist
Get whether each of the named virtual servers should decrypt SSL traffic.

Boolean[] getSSLDecrypt(
    String[] names
)
getSSLDecryptByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should decrypt SSL traffic. This is a location specific function, any action will operate on the specified location.

Boolean[] getSSLDecryptByLocation(
    String location
    String[] names
)

getSSLEllipticCurves( names ) throws ObjectDoesNotExist

Get the elliptic curve preference list for SSL connections to this virtual server

String[] getSSLEllipticCurves(
    String[] names
)

getSSLEllipticCurvesByLocation( location, names ) throws ObjectDoesNotExist

Get the elliptic curve preference list for SSL connections to this virtual server. This is a location specific function, any action will operate on the specified location.

String[] getSSLEllipticCurvesByLocation(
    String location
    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
)

getSSLExpectStartTLSByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should upgrade SMTP connections to SSL using the STARTTLS command. This is a location specific function, any action will operate on the specified location.

Boolean[] getSSLExpectStartTLSByLocation(
    String location
    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
)

getSSLHeadersByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should add HTTP headers to each request to show SSL connection parameters. This is a location specific function, any action will operate on the specified location.
Function Reference

VirtualServer

Boolean[] getSSLHeadersByLocation(String location, String[] names)

getSSLHonorFallbackSCSV( names ) throws ObjectDoesNotExist
Get whether Fallback SCSV is honored by this virtual server
VirtualServer.SSLHonorFallbackSCSV[] getSSLHonorFallbackSCSV(String[] names)

getSSLHonorFallbackSCSVByLocation( location, names ) throws ObjectDoesNotExist
Get whether Fallback SCSV is honored by this virtual server. This is a location specific function, any action will operate on the specified location.
VirtualServer.SSLHonorFallbackSCSV[] getSSLHonorFallbackSCSVByLocation(String location, 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.getLogConnectionFailures to control this configuration.
Boolean[] getSSLLogEnabled(String[] names)

getSSLNeverExpiringClientCertificateAuthorities( names ) throws ObjectDoesNotExist
Get the CAs for which any client certificate they validate is considered valid even if the client certificate’s expiration date has passed.
String[][] getSSLNeverExpiringClientCertificateAuthorities(String[] names)

getSSLNeverExpiringClientCertificateAuthoritiesByLocation( location, names ) throws ObjectDoesNotExist
Get the CAs for which any client certificate they validate is considered valid even if the client certificate’s expiration date has passed. This is a location specific function, any action will operate on the specified location.
String[][] getSSLNeverExpiringClientCertificateAuthoritiesByLocation(String location, String[] names)
getSSLNeverExpiringClientCertificateAuthoritiesDepth( names ) throws ObjectDoesNotExist

Get the number of certificates in a certificate chain beyond those listed as NeverExpiringClientCertificateAuthorities whose certificate expiry will not be checked.

Unsigned Integer[] getSSLNeverExpiringClientCertificateAuthoritiesDepth(
    String[] names
)

getSSLNeverExpiringClientCertificateAuthoritiesDepthByLocation( location, names ) throws ObjectDoesNotExist

Get the number of certificates in a certificate chain beyond those listed as NeverExpiringClientCertificateAuthorities whose certificate expiry will not be checked. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getSSLNeverExpiringClientCertificateAuthoritiesDepthByLocation(
    String location
    String[] names
)

getSSLOCSPDefaults( names ) throws ObjectDoesNotExist

Get the default OCSP responder settings for all client certificates.

VirtualServer.SSLOCSPIssuer[] getSSLOCSPDefaults(
    String[] names
)

getSSLOCSPDefaultsByLocation( location, names ) throws ObjectDoesNotExist

Get the default OCSP responder settings for all client certificates. This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLOCSPIssuer[] getSSLOCSPDefaultsByLocation(
    String location
    String[] names
)

getSSLOCSPIssuers( names ) throws ObjectDoesNotExist

Gets a list of mappings between Certificate Authorities and OCSP responder settings. Certificates issued by these authorities will be verified with OCSP using these settings.

VirtualServer.SSLOCSPIssuer[] getSSLOCSPIssuers(
    String[] names
)

getSSLOCSPIssuersByLocation( location, names ) throws ObjectDoesNotExist

Gets a list of mappings between Certificate Authorities and OCSP responder settings. Certificates issued by these authorities will be verified with OCSP using these settings. This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLOCSPIssuer[] getSSLOCSPIssuersByLocation(
    String location
    String[] names
)
getSSLOCSPMaxResponseAge( names ) throws ObjectDoesNotExist

Get the number of seconds for which an OCSP response is considered valid if it has not yet exceeded the time specified in the 'nextUpdate' field.

Unsigned Integer[] getSSLOCSPMaxResponseAge(
    String[] names
)

getSSLOCSPMaxResponseAgeByLocation( location, names ) throws ObjectDoesNotExist

Get the number of seconds for which an OCSP response is considered valid if it has not yet exceeded the time specified in the 'nextUpdate' field. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getSSLOCSPMaxResponseAgeByLocation(
    String location
    String[] names
)

getSSLOCSPStapling( names ) throws ObjectDoesNotExist

Get whether the traffic manager is allowed to provide OCSP responses for certificates as part of the handshake, if the client sends a TLS status_request extension in the ClientHello, and OCSP URIs are present in certificates used by this virtual server.

Boolean[] getSSLOCSPStapling(
    String[] names
)

getSSLOCSPStaplingByLocation( location, names ) throws ObjectDoesNotExist

Get whether the traffic manager is allowed to provide OCSP responses for certificates as part of the handshake, if the client sends a TLS status_request extension in the ClientHello, and OCSP URIs are present in certificates used by this virtual server. This is a location specific function, any action will operate on the specified location.

Boolean[] getSSLOCSPStaplingByLocation(
    String location
    String[] names
)

getSSLOCSPTimeTolerance( names ) throws ObjectDoesNotExist

Get the number of seconds outside the permitted range for which the 'thisUpdate' and 'nextUpdate' fields of an OCSP response are still considered valid.

Unsigned Integer[] getSSLOCSPTimeTolerance(
    String[] names
)

getSSLOCSPTimeToleranceByLocation( location, names ) throws ObjectDoesNotExist

Get the number of seconds outside the permitted range for which the 'thisUpdate' and 'nextUpdate' fields of an OCSP response are still considered valid. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getSSLOCSPTimeToleranceByLocation(
VirtualServer Function Reference

String location
String[] names
)

getSSLOCSPTimeout( names ) throws ObjectDoesNotExist

Get the number of seconds after which OCSP requests will be timed out
Unsigned Integer[] getSSLOCSPTimeout(
    String[] names
)

getSSLOCSPTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get the number of seconds after which OCSP requests will be timed out. This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getSSLOCSPTimeoutByLocation(
    String location
    String[] names
)

getSSLPreferSSLv3( names ) throws ObjectDoesNotExist

This method is now deprecated.
Boolean[] getSSLPreferSSLv3(
    String[] names
)

getSSLPreferSSLv3ByLocation( location, names ) throws ObjectDoesNotExist

This method is now deprecated. This is a location specific function, any action will operate on the specified location.
Boolean[] getSSLPreferSSLv3ByLocation(
    String location
    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
)

getSSLRequestClientCertModeByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should request (or require) an identifying certificate from each client. This is a location specific function, any action will operate on the specified location.
VirtualServer.SSLRequestClientCertMode[] getSSLRequestClientCertModeByLocation(
    String location
    String[] names
)
getSSLSendCloseAlerts( names ) throws ObjectDoesNotExist

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

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

getSSLSendCloseAlertsByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should send a close alert when initiating SSL socket disconnections. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLSendCloseAlertsByLocation(
    String location
    String[] names
)
```

getSSLSignatureAlgorithms( names ) throws ObjectDoesNotExist

Get the SSL signature algorithms preference list for SSL connections to this virtual server

```java
String[] getSSLSignatureAlgorithms(
    String[] names
)
```

getSSLSignatureAlgorithmsByLocation( location, names ) throws ObjectDoesNotExist

Get the SSL signature algorithms preference list for SSL connections to this virtual server. This is a location specific function, any action will operate on the specified location.

```java
String[] getSSLSignatureAlgorithmsByLocation(
    String location
    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.

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

getSSLSitesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
VirtualServer.SSLSite[][] getSSLSitesByLocation(
    String location
    String[] names
)
getSSLSitesEx( names ) throws ObjectDoesNotExist

Gets a list of mappings between destination addresses and the default certificate and alt certificates 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.SSLSiteAlt[][] getSSLSitesEx(
  String[] names
)

getSSLSitesExByLocation( location, names ) throws ObjectDoesNotExist

Gets a list of mappings between destination addresses and the default certificate and alt certificates 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. This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLSiteAlt[][] getSSLSitesExByLocation(
  String location
  String[] names
)

getSSLSupportSSL2( names ) throws ObjectDoesNotExist

This method is now deprecated.

VirtualServer.SSLSupportSSL2[] getSSLSupportSSL2(
  String[] names
)

getSSLSupportSSL2ByLocation( location, names ) throws ObjectDoesNotExist

This method is now deprecated. This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLSupportSSL2[] getSSLSupportSSL2ByLocation(
  String location
  String[] names
)

getSSLSupportSSL3( names ) throws ObjectDoesNotExist

Get whether SSLv3 is enabled for this virtual server

VirtualServer.SSLSupportSSL3[] getSSLSupportSSL3(
  String[] names
)

getSSLSupportSSL3ByLocation( location, names ) throws ObjectDoesNotExist

Get whether SSLv3 is enabled for this virtual server This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLSupportSSL3[] getSSLSupportSSL3ByLocation(
  String location
  String[] names
)
getSSLSupportTLS1( names ) throws ObjectDoesNotExist

Get whether TLSv1.0 is enabled for this virtual server

VirtualServer.SSLSupportTLS1[] getSSLSupportTLS1(
  String[] names
)

getSSLSupportTLS11( names ) throws ObjectDoesNotExist

Get whether TLSv1.1 is enabled for this virtual server

VirtualServer.SSLSupportTLS11[] getSSLSupportTLS11(
  String[] names
)

getSSLSupportTLS11ByLocation( location, names ) throws ObjectDoesNotExist

Get whether TLSv1.1 is enabled for this virtual server This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLSupportTLS11[] getSSLSupportTLS11ByLocation(
  String location
  String[] names
)

getSSLSupportTLS12( names ) throws ObjectDoesNotExist

Get whether TLSv1.2 is enabled for this virtual server

VirtualServer.SSLSupportTLS12[] getSSLSupportTLS12(
  String[] names
)

getSSLSupportTLS12ByLocation( location, names ) throws ObjectDoesNotExist

Get whether TLSv1.2 is enabled for this virtual server This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLSupportTLS12[] getSSLSupportTLS12ByLocation(
  String location
  String[] names
)

getSSLSupportTLS1ByLocation( location, names ) throws ObjectDoesNotExist

Get whether TLSv1.0 is enabled for this virtual server This is a location specific function, any action will operate on the specified location.

VirtualServer.SSLSupportTLS1[] getSSLSupportTLS1ByLocation(
  String location
  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
)
getSSLTrustMagicByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLTrustMagicByLocation(
    String location
    String[] names
)
```

getSSLUseOCSP( names ) throws ObjectDoesNotExist

Get whether or not the traffic manager should use OCSP to check the revocation status of client certificates

```java
Boolean[] getSSLUseOCSP(
    String[] names
)
```

getSSLUseOCSPByLocation( location, names ) throws ObjectDoesNotExist

Get whether or not the traffic manager should use OCSP to check the revocation status of client certificates. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLUseOCSPByLocation(
    String location
    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.

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

getServerfirstBannerByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
String[] getServerfirstBannerByLocation(
    String location
    String[] names
)
```

getServiceLevelMonitoring( names ) throws ObjectDoesNotExist

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

```java
String[] getServiceLevelMonitoring(
    String[] names
)
**Function Reference**

**VirtualServer**

**getServiceLevelMonitoringByLocation( location, names ) throws ObjectDoesNotExist**

Get the Service Level Monitoring class that each of the named virtual servers uses. This is a location specific function, any action will operate on the specified location.

```java
String[] getServiceLevelMonitoringByLocation(
    String location
    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.

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

**getSipTransactionTimeoutByLocation( location, names ) throws ObjectDoesNotExist**

Get the time after which an incomplete transaction should be discarded, in seconds, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getSipTransactionTimeoutByLocation(
    String location
    String[] names
)
```

**getStripXForwardedProtoHeader( names ) throws ObjectDoesNotExist**

Get Whether or not the virtual server should strip the \`X-Forwarded-Proto` header from incoming requests.

```java
Boolean[] getStripXForwardedProtoHeader(
    String[] names
)
```

**getStripXForwardedProtoHeaderByLocation( location, names ) throws ObjectDoesNotExist**

Get Whether or not the virtual server should strip the \`X-Forwarded-Proto` header from incoming requests. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getStripXForwardedProtoHeaderByLocation(
    String location
    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.

```java
Unsigned Integer[] getTimeout(
    String[] names
)
getTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get the time to wait for data on an already established connection, in seconds, for each of the named virtual
servers. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getTimeoutByLocation(
    String location
    String[] names
)

getTransactionExportBrief( names ) throws ObjectDoesNotExist

Get whether to export a restricted set of metadata about transactions processed by this virtual server. If
enabled, more verbose information such as client and server headers and request tracing events will be
omitted from the exported data.

Boolean[] getTransactionExportBrief(
    String[] names
)

getTransactionExportBriefByLocation( location, names ) throws ObjectDoesNotExist

Get whether to export a restricted set of metadata about transactions processed by this virtual server. If
enabled, more verbose information such as client and server headers and request tracing events will be
omitted from the exported data. This is a location specific function, any action will operate on the specified
location.

Boolean[] getTransactionExportBriefByLocation(
    String location
    String[] names
)

getTransactionExportEnabled( names ) throws ObjectDoesNotExist

Get whether to export metadata about transactions handled by this service to the globally configured
endpoint.

Boolean[] getTransactionExportEnabled(
    String[] names
)

getTransactionExportEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether to export metadata about transactions handled by this service to the globally configured
endpoint. This is a location specific function, any action will operate on the specified location.

Boolean[] getTransactionExportEnabledByLocation(
    String location
    String[] names
)

getTransactionExportHTTPHeaderBlacklist( names ) throws ObjectDoesNotExist

Get the set of HTTP header names for which corresponding values should be redacted from the metadata
exported by this virtual server from transaction export.

String[][] getTransactionExportHTTPHeaderBlacklist(
    String[] names
)
getTransactionExportHTTPHeaderBlacklistByLocation(location, names) throws ObjectDoesNotExist

Get the set of HTTP header names for which corresponding values should be redacted from the metadata exported by this virtual server from transaction export. This is a location specific function, any action will operate on the specified location.

```java
String[][] getTransactionExportHTTPHeaderBlacklistByLocation(
    String location
    String[] names
)
```

getTransactionExportHiRes(names) throws ObjectDoesNotExist

Get whether the transaction processing timeline included in the metadata export is recorded with a high, microsecond, resolution. If set to No, timestamps will be recorded with a resolution of milliseconds.

```java
Boolean[] getTransactionExportHiRes(
    String[] names
)
```

getTransactionExportHiResByLocation(location, names) throws ObjectDoesNotExist

Get whether the transaction processing timeline included in the metadata export is recorded with a high, microsecond, resolution. If set to No, timestamps will be recorded with a resolution of milliseconds. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getTransactionExportHiResByLocation(
    String location
    String[] names
)
```

getTransparent(names) throws ObjectDoesNotExist

Get whether or not bound sockets should be configured for transparent proxying.

```java
Boolean[] getTransparent(
    String[] names
)
```

getTransparentByLocation(location, names) throws ObjectDoesNotExist

Get whether or not bound sockets should be configured for transparent proxying. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getTransparentByLocation(
    String location
    String[] names
)
```

getUDPEndTransaction(names) throws ObjectDoesNotExist

Get when the traffic manager should consider a UDP transaction to have ended.

```java
VirtualServer.UDPEndTransaction[] getUDPEndTransaction(
    String[] names
)
```
getUDPEndTransactionByLocation( location, names ) throws ObjectDoesNotExist

Get when the traffic manager should consider a UDP transaction to have ended. This is a location specific function, any action will operate on the specified location.

VirtualServer.UDPEndTransaction[] getUDPEndTransactionByLocation(
    String location
    String[] names
)

getUDPEndpointPersistence( names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should send UDP datagrams received from the same IP address and port to the same pool node if they match an existing UDP session. Sessions are defined by the protocol being handled, for example SIP datagrams are grouped based on the value of the Call-ID header.

Boolean[] getUDPEndpointPersistence(
    String[] names
)

getUDPEndpointPersistenceByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should send UDP datagrams received from the same IP address and port to the same pool node if they match an existing UDP session. Sessions are defined by the protocol being handled, for example SIP datagrams are grouped based on the value of the Call-ID header. This is a location specific function, any action will operate on the specified location.

Boolean[] getUDPEndpointPersistenceByLocation(
    String location
    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
)

getUDPResponseDatagramsExpectedByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Integer[] getUDPResponseDatagramsExpectedByLocation(
    String location
    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
)

getUDPTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getUDPTimeoutByLocation(
   String location
   String[] names
)

getAddressNagle( names ) throws ObjectDoesNotExist

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

Boolean[] getUseNagle(
   String[] names
)

getAddressNagleByLocation( location, names ) throws ObjectDoesNotExist

Get whether Nagle's algorithm should be used for TCP connections. This is a location specific function, any action will operate on the specified location.

Boolean[] getUseNagleByLocation(
   String location
   String[] names
)

getVirtualServerNames()

Gets the names of all the configured virtual servers.

String[] getVirtualServerNames()

getAddressCacheControlOut( names ) throws ObjectDoesNotExist

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

String[] getWebcacheControlOut(
   String[] names
)

getAddressCacheControlOutByLocation( location, names ) throws ObjectDoesNotExist

Get the Cache-Control header that should be sent with cached HTTP responses. This is a location specific function, any action will operate on the specified location.

String[] getWebcacheControlOutByLocation(
   String location
   String[] names
)
getWebcacheEnabled( names ) throws ObjectDoesNotExist

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

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

getWebcacheEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named virtual servers should attempt to cache web server responses. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getWebcacheEnabledByLocation(
    String location
    String[] names
)
```

getWebcacheErrorpageTime( names ) throws ObjectDoesNotExist

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

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

getWebcacheErrorpageTimeByLocation( location, names ) throws ObjectDoesNotExist

Get the time periods that each of the named virtual servers should cache error pages for. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getWebcacheErrorpageTimeByLocation(
    String location
    String[] names
)
```

getWebcacheRefreshTime( names ) throws ObjectDoesNotExist

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

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

getWebcacheRefreshTimeByLocation( location, names ) throws ObjectDoesNotExist

Get the time periods that each of the named virtual servers should consider re-fetching cached pages in. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getWebcacheRefreshTimeByLocation(
    String location
    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
)

getWebcacheTimeByLocation( location, names ) throws ObjectDoesNotExist

Get the time periods that each of the named virtual servers should cache web pages for. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getWebcacheTimeByLocation(
    String location
    String[] names
)

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

For each of the named virtual servers, remove rules from the list that are run on the completion of a transaction.

void removeCompletionRules(
    String[] names
    String[][] rules
)

removeCompletionRulesByLocation( location, names, rules ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, remove rules from the list that are run on the completion of a transaction. This is a location specific function, any action will operate on the specified location.

void removeCompletionRulesByLocation(
    String location
    String[] names
    String[][] rules
)

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
)

removeCompressionMIMETypesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError

For each named virtual server, remove new MIME types from the list of types to compress. This is a location specific function, any action will operate on the specified location.

void removeCompressionMIMETypesByLocation(
    String location
    String[] names
String[][] values

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

Remove Space separated list of DNS zones

void removeDNSZones{
    String[] names
    String[][] values
}

removeDNSZonesByLocation(location, names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Remove Space separated list of DNS zones This is a location specific function, any action will operate on the specified location.

void removeDNSZonesByLocation{
    String location
    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
}

removeFTPPortRangeByLocation(location, names) throws ObjectDoesNotExist, DeploymentError

Allow FTP connections to use any free ports, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void removeFTPPortRangeByLocation{
    String location
    String[] names
}

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

Remove GLB Services used by the virtual server

void removeGLBServices{
    String[] names
    String[][] values
}

removeGLBServicesByLocation(location, names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Remove GLB Services used by the virtual server This is a location specific function, any action will operate on the specified location.
Function Reference

void removeGLBServicesByLocation(
    String location
    String[] names
    String[][] values
)

removeHTTP2HeadersIndexBlacklist( names, values ) throws ObjectDoesNotExist, DeploymentError

Remove a list of header names that should never be compressed using indexing.

void removeHTTP2HeadersIndexBlacklist(
    String[] names
    String[][] values
)

removeHTTP2HeadersIndexBlacklistByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError

Remove a list of header names that should never be compressed using indexing. This is a location specific function, any action will operate on the specified location.

void removeHTTP2HeadersIndexBlacklistByLocation(
    String location
    String[] names
    String[][] values
)

removeHTTP2HeadersIndexWhitelist( names, values ) throws ObjectDoesNotExist, DeploymentError

Remove a list of header names that can be compressed using indexing when the default is to never index.

void removeHTTP2HeadersIndexWhitelist(
    String[] names
    String[][] values
)

removeHTTP2HeadersIndexWhitelistByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError

Remove a list of header names that can be compressed using indexing when the default is to never index. This is a location specific function, any action will operate on the specified location.

void removeHTTP2HeadersIndexWhitelistByLocation(
    String location
    String[] names
    String[][] values
)

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
)
removeRTSPPortRangeByLocation( location, names ) throws ObjectDoesNotExist, DeploymentError

Allow any free ports to be used for RTSP connections, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void removeRTSPPortRangeByLocation(
    String location
    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.

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

removeResponseRulesByLocation( location, names, rules ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, remove rules from the list that are run on server responses. This is a location specific function, any action will operate on the specified location.

```java
void removeResponseRulesByLocation(
    String location
    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.

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

removeRulesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void removeRulesByLocation(
    String location
    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.

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

**removeSIPPortRangeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void removeSIPPortRangeByLocation(
    String location
    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
)

**removeSSLClientCertificateAuthoritiesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError**

Remove certificate authorities for validating client certificates for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void removeSSLClientCertificateAuthoritiesByLocation(
    String location
    String[] names
    String[][] values
)

**removeSSLNeverExpiringClientCertificateAuthorities( names, values ) throws ObjectDoesNotExist, DeploymentError**

Remove CAs for which any client certificate they validate is considered valid even if the client certificate's expiration date has passed, for each of the named virtual servers.

void removeSSLNeverExpiringClientCertificateAuthorities(
    String[] names
    String[][] values
)

**removeSSLNeverExpiringClientCertificateAuthoritiesByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError**

Remove CAs for which any client certificate they validate is considered valid even if the client certificate's expiration date has passed, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void removeSSLNeverExpiringClientCertificateAuthoritiesByLocation(
    String location
    String[] names
)
String[][] values

```java
void removeSSLOCSPIssuers(String[] names, String[][] cas) throws ObjectDoesNotExist, DeploymentError
```

Removes mappings between OCSP responder settings for the specified Certificate authorities

```java
void removeSSLOCSPIssuersByLocation(String location, String[] names, String[][] cas) throws ObjectDoesNotExist, DeploymentError
```

Removes mappings between OCSP responder settings for the specified Certificate authorities This is a location specific function, any action will operate on the specified location.

```java
void removeTransactionExportHTTPHeaderBlacklist(String[] names, String[][] values) throws ObjectDoesNotExist, DeploymentError
```

Remove the set of HTTP header names for which corresponding values should be redacted from the metadata exported by this virtual server from transaction export.

```java
void removeTransactionExportHTTPHeaderBlacklistByLocation(String location, String[] names, String[][] values) throws ObjectDoesNotExist, DeploymentError
```

Remove the set of HTTP header names for which corresponding values should be redacted from the metadata exported by this virtual server from transaction export. This is a location specific function, any action will operate on the specified location.

```java
void renameVirtualServer(String[] names, String[] new_names) throws ObjectDoesNotExist, InvalidObjectName, DeploymentError, InvalidOperation
```

Rename each of the named virtual servers.
**setAddClusterClientIPHeader( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether an '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.

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

**setAddClusterClientIPHeaderByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether an '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. This is a location specific function, any action will operate on the specified location.

```java
void setAddClusterClientIPHeaderByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setAddXForwardedForHeader( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether the remote client's IP address should be appended to the X-Forwarded-For header. The 'X-Forwarded-For' header contains the client's IP address.

```java
void setAddXForwardedForHeader(
    String[] names
    Boolean[] values
)
```

**setAddXForwardedForHeaderByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether the remote client's IP address should be appended to the X-Forwarded-For header. The 'X-Forwarded-For' header contains the client's IP address. This is a location specific function, any action will operate on the specified location.

```java
void setAddXForwardedForHeaderByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setAddXForwardedProtoHeader( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether an 'X-Forwarded-Proto' header should be added to each HTTP request, for each of the named virtual servers. The 'X-Forwarded-Proto' header contains the protocol the client used to connect to the traffic manager.

```java
void setAddXForwardedProtoHeader(
    String[] names
    Boolean[] values
)
```
setAddXForwardedProtoHeaderByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether an 'X-Forwarded-Proto' header should be added to each HTTP request, for each of the named virtual servers. The 'X-Forwarded-Proto' header contains the protocol the client used to connect to the traffic manager. This is a location specific function, any action will operate on the specified location.

```java
void setAddXForwardedProtoHeaderByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

For each of the named virtual servers, enable or disable the Brocade Virtual Web Application Firewall.

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

setApplicationFirewallEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, LicenseError

For each of the named virtual servers, enable or disable the Brocade Virtual Web Application Firewall. This is a location specific function, any action will operate on the specified location.

```java
void setApplicationFirewallEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set whether each of the named virtual servers should optimize web content.

```java
void setAptimizerEnabled(
    String[] names
    Boolean[] values
)
```

setAptimizerEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should optimize web content. This is a location specific function, any action will operate on the specified location.

```java
void setAptimizerEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
**setAutodetectUpgradeHeaders( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether the traffic manager should check for HTTP responses that confirm an upgrade to the WebSockets protocol and automatically stop any protocol-specific processing for that connection when detected.

```java
void setAutodetectUpgradeHeaders(
    String[] names
    Boolean[] values
)
```

**setAutodetectUpgradeHeadersByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether the traffic manager should check for HTTP responses that confirm an upgrade to the WebSockets protocol and automatically stop any protocol-specific processing for that connection when detected. This is a location specific function, any action will operate on the specified location.

```java
void setAutodetectUpgradeHeadersByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

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

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

**setBandwidthClassByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the Bandwidth Class that each of the named virtual servers uses. This is a location specific function, any action will operate on the specified location.

```java
void setBandwidthClassByLocation(
    String location
    String[] names
    String[] values
)
```

**setBypassDataPlaneAcceleration( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether this service should, where possible, bypass data plane acceleration mechanisms.

```java
void setBypassDataPlaneAcceleration(
    String[] names
    Boolean[] values
)
```
**VirtualServer Function Reference**

**setBypassDataPlaneAccelerationByLocation**

Set whether this service should, where possible, bypass data plane acceleration mechanisms. This is a location specific function, any action will operate on the specified location.

```java
void setBypassDataPlaneAccelerationByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

**setCloseWithRst**

Set whether connections from clients should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state.

```java
void setCloseWithRst(
    String[] names,
    Boolean[] values
)
```

**setCloseWithRstByLocation**

Set whether connections from clients should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state. This is a location specific function, any action will operate on the specified location.

```java
void setCloseWithRstByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

**setCompletionRules**

Set the rules that are run on the completion of a transaction for each of the named virtual servers.

```java
void setCompletionRules(
    String[] names,
    VirtualServer.Rule[][] rules
)
```

**setCompletionRulesByLocation**

Set the rules that are run on the completion of a transaction for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setCompletionRulesByLocation(
    String location,
    String[] names,
    VirtualServer.Rule[][] rules
)
```
**setCompressUnknownSize( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

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

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

**setCompressUnknownSizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether each of the named virtual servers should compress documents with no given size. This is a location specific function, any action will operate on the specified location.

```java
void setCompressUnknownSizeByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setCompressionETagRewrite( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set how the ETag header should be manipulated when compressing content.

```java
void setCompressionETagRewrite(
    String[] names
    VirtualServer.CompressionETagRewrite[] values
)
```

**setCompressionETagRewriteByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set how the ETag header should be manipulated when compressing content. This is a location specific function, any action will operate on the specified location.

```java
void setCompressionETagRewriteByLocation(
    String location
    String[] names
    VirtualServer.CompressionETagRewrite[] 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.

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

**setCompressionEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether each of the named virtual servers should compress web pages before sending to the client. This is a location specific function, any action will operate on the specified location.

```java
void setCompressionEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```
void setCompressionEnabledByLocation(
    String location
    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
)

_setCompressionLevelByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the gzip compression level, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void setCompressionLevelByLocation(
    String location
    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
)

_setCompressionMIMETypesByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the list of MIME types to compress, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void setCompressionMIMETypesByLocation(
    String location
    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
)
setCompressionMaxSizeByLocation( location, 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'. This is a location specific function, any action will operate on the specified location.

```java
void setCompressionMaxSizeByLocation(
    String location
    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.

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

setCompressionMinSizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the minimum document size to compress, in bytes, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setCompressionMinSizeByLocation(
    String location
    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.

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

setConnectTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setConnectTimeoutByLocation(
    String location
    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.

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

**setCookieDomainRewriteModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setCookieDomainRewriteModeByLocation(
    String location
    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.

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

**setCookieNamedDomainByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the domain to use when rewriting cookie domains, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setCookieNamedDomainByLocation(
    String location
    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.

```java
void setCookiePathRewrite(
    String[] names
    VirtualServer.RegexReplacement[] values
)
setCookiePathRewriteByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setCookiePathRewriteByLocation(
    String location
    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.

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

setCookieSecureFlagRewriteModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setCookieSecureFlagRewriteModeByLocation(
    String location
    String[] names
    VirtualServer.CookieSecureFlagRewriteMode[] values
)
```

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

Set whether use of EDNS client subet option is enabled

```java
void setDNSEdnsClientSubnet(
    String[] names
    Boolean[] values
)
```

setDNSEdnsClientSubnetByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether use of EDNS client subnet option is enabled This is a location specific function, any action will operate on the specified location.

```java
void setDNSEdnsClientSubnetByLocation(
    String location
    String[] names
    Boolean[] values
)
```
setDNSEdnsUdpsize( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set EDNS UDP size advertised in responses

```java
void setDNSEdnsUdpsize(
    String[] names
    Unsigned Integer[] values
)
```

setDNSEdnsUdpsizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set EDNS UDP size advertised in responses This is a location specific function, any action will operate on the specified location.

```java
void setDNSEdnsUdpsizeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set Maximum UDP answer size

```java
void setDNSMaxUdpsize(
    String[] names
    Unsigned Integer[] values
)
```

setDNSMaxUdpsizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set Maximum UDP answer size This is a location specific function, any action will operate on the specified location.

```java
void setDNSMaxUdpsizeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set order of records in a DNS response

```java
void setDNSRecordsetOrder(
    String[] names
    VirtualServer.DNSRecordsetOrder[] values
)
```

setDNSRecordsetOrderByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set order of records in a DNS response This is a location specific function, any action will operate on the specified location.

```java
void setDNSRecordsetOrderByLocation(
    String location
    String[] names
    VirtualServer.DNSRecordsetOrder[] values
)
```
void setDNSRecordsetOrderByLocation{
    String location
    String[] names
    VirtualServer.DNSRecordsetOrder[] values
}

setDNSVerbose( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set whether built-in DNS server should log verbose messages
void setDNSVerbose{
    String[] names
    Boolean[] values
}

setDNSVerboseByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set whether built-in DNS server should log verbose messages This is a location specific function, any action will operate on the specified location.
void setDNSVerboseByLocation{
    String location
    String[] names
    Boolean[] values
}

setDNSZones( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set Space separated list of DNS zones
void setDNSZones{
    String[] names
    String[][] values
}

setDNSZonesByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set Space separated list of DNS zones This is a location specific function, any action will operate on the specified location.
void setDNSZonesByLocation{
    String location
    String[] names
    String[][] 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
}
**setDefaultPoolByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the default Pool that traffic is sent to for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setDefaultPoolByLocation(
    String location
    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).

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

**setEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether each of the named virtual servers is enabled (i.e. serving traffic). This is a location specific function, any action will operate on the specified location.

```java
void setEnabledByLocation(
    String location
    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.

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

**setErrorFileByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setErrorFileByLocation(
    String location
    String[] names
    String[] values
)
```
Function Reference

VirtualServer

**setFTPDataSourcePort**

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.

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

**setFTPDataSourcePortByLocation**

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. This is a location specific function, any action will operate on the specified location.

```java
void setFTPDataSourcePortByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setFTPForceClientSecure**

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.

```java
void setFTPForceClientSecure(
    String[] names
    Boolean[] values
)
```

**setFTPForceClientSecureByLocation**

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. This is a location specific function, any action will operate on the specified location.

```java
void setFTPForceClientSecureByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setFTPForceServerSecure**

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.

```java
void setFTPForceServerSecure(
    String[] names
    Boolean[] values
)
```
setFTPForceServerSecureByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setFTPForceServerSecureByLocation(
    String location
    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.

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

setFTPPortRangeByLocation( location, names, range ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port range used for FTP data connections for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setFTPPortRangeByLocation(
    String location
    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

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

setFTPSSLDataByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setFTPSSLDataByLocation(
    String location
    String[] names
    Boolean[] values
)
```
Function Reference

VirtualServer

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

Set GLB Services used by the virtual server

```java
void setGLBServices{
  String[] names
  String[][] values
}
```

setGLBServicesByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set GLB Services used by the virtual server This is a location specific function, any action will operate on the specified location.

```java
void setGLBServicesByLocation{
  String location
  String[] names
  String[][] values
}
```

setHTTP2ConnectTimeout( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time to wait for a request on a new HTTP/2 connection, in seconds. If no request is received in this time, the connection will be closed. This setting overrides 'connect_timeout', and uses the value of 'connect_timeout' if set to zero.

```java
void setHTTP2ConnectTimeout{
  String[] names
  Unsigned Integer[] values
}
```

setHTTP2ConnectTimeoutByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time to wait for a request on a new HTTP/2 connection, in seconds. If no request is received in this time, the connection will be closed. This setting overrides 'connect_timeout', and uses the value of 'connect_timeout' if set to zero. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2ConnectTimeoutByLocation{
  String location
  String[] names
  Unsigned Integer[] values
}
```

setHTTP2DataFrameSize( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the preferred HTTP/2 data frame size

```java
void setHTTP2DataFrameSize{
  String[] names
  Unsigned Integer[] values
}
```
**setHTTP2DataFrameSizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the preferred HTTP/2 data frame size. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2DataFrameSizeByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```

**setHTTP2Enabled( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set allows the HTTP/2 protocol to be used.

```java
void setHTTP2Enabled(
    String[] names,
    Boolean[] values
)
```

**setHTTP2EnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set allows the HTTP/2 protocol to be used. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2EnabledByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

**setHTTP2HeaderTableSize( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the amount of memory allowed for header compression.

```java
void setHTTP2HeaderTableSize(
    String[] names,
    Unsigned Integer[] values
)
```

**setHTTP2HeaderTableSizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the amount of memory allowed for header compression. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2HeaderTableSizeByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```
setHTTP2HeadersIndexBlacklist( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set a list of header names that should never be compressed using indexing.

```java
void setHTTP2HeadersIndexBlacklist(
    String[] names
    String[][] values
)
```

setHTTP2HeadersIndexBlacklistByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set a list of header names that should never be compressed using indexing. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2HeadersIndexBlacklistByLocation(
    String location
    String[] names
    String[][] values
)
```

setHTTP2HeadersIndexDefault( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether all HTTP/2 headers should be compressed using indexing, except those specified in the blacklist, or whether all HTTP/2 headers should not be compressed using indexing, except those specified in the whitelist.

```java
void setHTTP2HeadersIndexDefault(
    String[] names
    Boolean[] values
)
```

setHTTP2HeadersIndexDefaultByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether all HTTP/2 headers should be compressed using indexing, except those specified in the blacklist, or whether all HTTP/2 headers should not be compressed using indexing, except those specified in the whitelist. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2HeadersIndexDefaultByLocation(
    String location
    String[] names
    Boolean[] values
)
```

setHTTP2HeadersIndexWhitelist( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set a list of header names that can be compressed using indexing when the default is to never index.

```java
void setHTTP2HeadersIndexWhitelist(
    String[] names
    String[][] values
)
**setHTTP2HeadersIndexWhitelistByLocation**

```java
void setHTTP2HeadersIndexWhitelistByLocation(
    String location,
    String[] names,
    String[][] values
)
```

Sets a list of header names that can be compressed using indexing when the default is to never index. This is a location specific function, any action will operate on the specified location.

**setHTTP2HeadersSizeLimit**

```java
void setHTTP2HeadersSizeLimit(
    String[] names,
    Unsigned Integer[] values
)
```

Sets the maximum size, in bytes, of decompressed headers for an HTTP/2 request.

**setHTTP2HeadersSizeLimitByLocation**

```java
void setHTTP2HeadersSizeLimitByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```

Sets the maximum size, in bytes, of decompressed headers for an HTTP/2 request. This is a location specific function, any action will operate on the specified location.

**setHTTP2IdleTimeoutNoStreams**

```java
void setHTTP2IdleTimeoutNoStreams(
    String[] names,
    Unsigned Integer[] values
)
```

Sets the time to wait for an HTTP/2 request on a previously used HTTP/2 connection with no open streams.

**setHTTP2IdleTimeoutNoStreamsByLocation**

```java
void setHTTP2IdleTimeoutNoStreamsByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```

Sets the time to wait for an HTTP/2 request on a previously used HTTP/2 connection with no open streams. This is a location specific function, any action will operate on the specified location.
setHTTP2IdleTimeoutOpenStreams( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time to wait for data on an HTTP/2 connection with open streams that haven't sent data recently

```java
void setHTTP2IdleTimeoutOpenStreams(
    String[] names
    Unsigned Integer[] values
)
```

setHTTP2IdleTimeoutOpenStreamsByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time to wait for data on an HTTP/2 connection with open streams that haven't sent data recently This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2IdleTimeoutOpenStreamsByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

setHTTP2MaxConcurrentStreams( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of concurrent streams allowed

```java
void setHTTP2MaxConcurrentStreams(
    String[] names
    Unsigned Integer[] values
)
```

setHTTP2MaxConcurrentStreamsByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of concurrent streams allowed This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2MaxConcurrentStreamsByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

setHTTP2MaxFrameSize( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the maximum HTTP/2 frame size

```java
void setHTTP2MaxFrameSize(
    String[] names
    Unsigned Integer[] values
)
```

setHTTP2MaxFrameSizeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the maximum HTTP/2 frame size This is a location specific function, any action will operate on the specified location.
void setHTTP2MaxFrameSizeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

setHTTP2MaxHeaderPadding( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set the maximum size, in bytes, of random-length padding to add to HTTP/2 header frames

void setHTTP2MaxHeaderPadding(
    String[] names
    Unsigned Integer[] values
)

setHTTP2MaxHeaderPaddingByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set the maximum size, in bytes, of random-length padding to add to HTTP/2 header frames. This is a location specific function, any action will operate on the specified location.

void setHTTP2MaxHeaderPaddingByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

setHTTP2MergeCookieHeaders( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set whether Cookie headers received from an HTTP/2 client should be merged into a single Cookie header before forwarding to an HTTP/1.1 server.

void setHTTP2MergeCookieHeaders(
    String[] names
    Boolean[] values
)

setHTTP2MergeCookieHeadersByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set whether Cookie headers received from an HTTP/2 client should be merged into a single Cookie header before forwarding to an HTTP/1.1 server. This is a location specific function, any action will operate on the specified location.

void setHTTP2MergeCookieHeadersByLocation(
    String location
    String[] names
    Boolean[] values
)

setHTTP2StreamWindowSize( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set the flow control window size

void setHTTP2StreamWindowSize(
    String[] names
    Unsigned Integer[] values
**Function Reference**

VirtualServer

```java
setHTTP2StreamWindowSizeByLocation(location, names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the flow control window size. This is a location specific function, any action will operate on the specified location.

```java
void setHTTP2StreamWindowSizeByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```

**setHttpChunkOverheadForwarding( names, values )** throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set how to handle forwarding of data that is pure HTTP chunking overhead.

```java
void setHttpChunkOverheadForwarding(
    String[] names,
    VirtualServer.HttpChunkOverheadForwarding[] values
)
```

**setHttpChunkOverheadForwardingByLocation( location, names, values )** throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set how to handle forwarding of data that is pure HTTP chunking overhead. This is a location specific function, any action will operate on the specified location.

```java
void setHttpChunkOverheadForwardingByLocation(
    String location,
    String[] names,
    VirtualServer.HttpChunkOverheadForwarding[] values
)
```

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

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

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

**setKeepaliveByLocation( location, names, values )** throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should allow clients to maintain keepalive connections. This is a location specific function, any action will operate on the specified location.

```java
void setKeepaliveByLocation(
    String location,
    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.

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

**setKeepaliveTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setKeepaliveTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setKerberosProtocolTransitionEnabled( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Sets whether each of the named virtual servers should use Kerberos Protocol Transition.

```java
void setKerberosProtocolTransitionEnabled(
    String[] names
    Boolean[] values
)
```

**setKerberosProtocolTransitionEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Sets whether each of the named virtual servers should use Kerberos Protocol Transition. This is a location specific function, any action will operate on the specified location.

```java
void setKerberosProtocolTransitionEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setKerberosProtocolTransitionPrincipal( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the Kerberos principal that each of the named virtual servers uses to perform Kerberos Protocol Transition.

```java
void setKerberosProtocolTransitionPrincipal(
    String[] names
    String[] values
)
```
setKerberosProtocolTransitionPrincipalByLocation( location, names, values )
throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Kerberos principal that each of the named virtual servers uses to perform Kerberos Protocol Transition. This is a location specific function, any action will operate on the specified location.

```java
void setKerberosProtocolTransitionPrincipalByLocation(
    String location,
    String[] names,
    String[] values
)
```

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

Set the Kerberos principal name of the service that each of the named virtual servers targets.

```java
void setKerberosProtocolTransitionTarget(
    String[] names,
    String[] values
)
```

setKerberosProtocolTransitionTargetByLocation( location, names, values )
throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Kerberos principal name of the service that each of the named virtual servers targets. This is a location specific function, any action will operate on the specified location.

```java
void setKerberosProtocolTransitionTargetByLocation(
    String location,
    String[] names,
    String[] values
)
```

setL4AccelRSTOnServiceFailure( names, values )
throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server should send a TCP RST packet or ICMP error message if a service is unavailable, or if an established connection to a node fails.

```java
void setL4AccelRSTOnServiceFailure(
    String[] names,
    Boolean[] values
)
```

setL4AccelRSTOnServiceFailureByLocation( location, names, values )
throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server should send a TCP RST packet or ICMP error message if a service is unavailable, or if an established connection to a node fails. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelRSTOnServiceFailureByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```
setL4AccelServiceIPSNAT( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether or not backend connections should be configured to use the ingress service IP as the source IP for the back-end connection when Source NAT is enabled for the pool used by the service.

```java
void setL4AccelServiceIPSNAT(
    String[] names
    Boolean[] values
)
```

setL4AccelServiceIPSNATByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether or not backend connections should be configured to use the ingress service IP as the source IP for the back-end connection when Source NAT is enabled for the pool used by the service. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelServiceIPSNATByLocation(
    String location
    String[] names
    Boolean[] values
)
```

setL4AccelStateSync( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the state of active connections will be synchronized across the cluster for L4Accel services, such that connections will persist in the event of a failover.

```java
void setL4AccelStateSync(
    String[] names
    Boolean[] values
)
```

setL4AccelStateSyncByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the state of active connections will be synchronized across the cluster for L4Accel services, such that connections will persist in the event of a failover. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelStateSyncByLocation(
    String location
    String[] names
    Boolean[] values
)
```

setL4AccelTCPMaxSegmentLifetime( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the maximum segment lifetime, in seconds, of a TCP segment being handled by the traffic manager. This setting determines for how long information about a connection will be retained after receiving a two-way FIN or RST.

```java
void setL4AccelTCPMaxSegmentLifetime(
    String[] names
    Unsigned Integer[] values
)
```
virtualserver

setL4AccelTCPMaxSegmentLifetimeByLocation(location, names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the maximum segment lifetime, in seconds, of a TCP segment being handled by the traffic manager. This setting determines for how long information about a connection will be retained after receiving a two-way FIN or RST. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelTCPMaxSegmentLifetimeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

setL4AccelTimeout(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of seconds after which a connection will be closed if no further packets have been received on it.

```java
void setL4AccelTimeout(
    String[] names
    Unsigned Integer[] values
)
```

setL4AccelTimeoutByLocation(location, names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of seconds after which a connection will be closed if no further packets have been received on it. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

setL4AccelUDPCountRequests(names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether a connection should be closed when the number of UDP response datagrams received from the server is equal to the number of request datagrams that have been sent by the client. If set to No the connection will be closed after the first response has been received from the server. This setting takes precedence over l4accel!optimized_aging setting.

```java
void setL4AccelUDPCountRequests(
    String[] names
    Boolean[] values
)
```

setL4AccelUDPCountRequestsByLocation(location, names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether a connection should be closed when the number of UDP response datagrams received from the server is equal to the number of request datagrams that have been sent by the client. If set to No the connection will be closed after the first response has been received from the server. This setting takes precedence over l4accel!optimized_aging setting. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelUDPCountRequestsByLocation(
```
VirtualServer Function Reference

String location
String[] names
Boolean[] 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
)
```

```
setListenAddressesByLocation( location, names, addresses ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
```

Set the specific IP addresses and hostnames for each named virtual server to listen on. This is a location specific function, any action will operate on the specified location.

```
void setListenAddressesByLocation(
    String location
    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
)
```

```
setListenOnAllAddressesByLocation( location, names ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
```

Make each of the named virtual servers listen on all IP addresses. This is a location specific function, any action will operate on the specified location.

```
void setListenOnAllAddressesByLocation(
    String location
    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
)
```
setListenTrafficIPGroupsByLocation( location, names, groups ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, set specific Traffic IP Groups for it to listen on. This is a location specific function, any action will operate on the specified location.

```java
void setListenTrafficIPGroupsByLocation(
    String location
    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.

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

setLocationDefaultRewriteModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setLocationDefaultRewriteModeByLocation(
    String location
    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.

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

setLocationRewriteByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

For each of the named virtual servers, set the regex, and replacement for rewriting any 'Location' headers. This is a location specific function, any action will operate on the specified location.

```java
void setLocationRewriteByLocation(
    String location
    String[] names
    VirtualServer.RegexReplacement[] values
)
```
setLogClientConnectionFailures( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log client connection failures.

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

setLogClientConnectionFailuresByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log client connection failures. This is a location specific function, any action will operate on the specified location.

```java
void setLogClientConnectionFailuresByLocation(
    String location
    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.

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

setLogEnabledByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setLogEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

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

```java
void setLogFileFilename(
    String[] names
    String[] values
)
```

setLogFileFilenameByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the name of the file used to store request logs, for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.
void setLogFilenameByLocation(
    String location
    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
)

setLogFormatByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the log file format for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void setLogFormatByLocation(
    String location
    String[] names
    String[] values
)

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

Set whether the virtual server will log ssl failures.

void setLogSSLFailures(
    String[] names
    Boolean[] values
)

setLogSSLFailuresByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log ssl failures. This is a location specific function, any action will operate on the specified location.

void setLogSSLFailuresByLocation(
    String location
    String[] names
    Boolean[] values
)

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

Set whether to log all connections by default, or log no connections by default.

void setLogSaveAll(
    String[] names
    Boolean[] values
)
setLogSaveAllByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether to log all connections by default, or log no connections by default. This is a location specific function, any action will operate on the specified location.

```java
void setLogSaveAllByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set whether the virtual server will log server connection failures.

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

setLogServerConnectionFailuresByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log server connection failures. This is a location specific function, any action will operate on the specified location.

```java
void setLogServerConnectionFailuresByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set whether the virtual server will log session persistence events.

```java
void setLogSessionPersistenceVerbose(
    String[] names
    Boolean[] values
)
```

setLogSessionPersistenceVerboseByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the virtual server will log session persistence events. This is a location specific function, any action will operate on the specified location.

```java
void setLogSessionPersistenceVerboseByLocation(
    String location
    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.

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

setMIMEAutoDetectByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setMIMEAutoDetectByLocation(
    String location
    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.

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

setMIMEDefaultTypeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setMIMEDefaultTypeByLocation(
    String location
    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.

```java
void setMaxClientBuffer(
    String[] names
    Unsigned Integer[] values
)
```
setMaxClientBufferByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setMaxClientBufferByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set the maximum number of concurrent TCP connections this virtual server will accept. A value of 0 will allow unlimited concurrent TCP connections to this virtual server.

```java
void setMaxConcurrentConnections(
    String[] names
    Unsigned Integer[] values
)
```

setMaxConcurrentConnectionsByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the maximum number of concurrent TCP connections this virtual server will accept. A value of 0 will allow unlimited concurrent TCP connections to this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setMaxConcurrentConnectionsByLocation(
    String location
    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.

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

setMaxServerBufferByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setMaxServerBufferByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```
setMaxTransactionDuration( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the total amount of time a transaction can take, zero means forever.

```java
void setMaxTransactionDuration(
    String[] names
    Unsigned Integer[] values
)
```

setMaxTransactionDurationByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the total amount of time a transaction can take, zero means forever. This is a location specific function, any action will operate on the specified location.

```java
void setMaxTransactionDurationByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set the note for each of the named virtual servers.

```java
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.

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

setPortByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port that each of the named virtual servers listens on for incoming connections. This is a location specific function, any action will operate on the specified location.

```java
void setPortByLocation(
    String location
    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.

```java
void setProtection(
```
setProtectionByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Service Protection Settings that are used to protect each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setProtectionByLocation(
    String location
    String[] names
    String[] values
)
```

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

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

```java
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 the traffic manager is responding to the request itself, enabling this setting will cause the traffic manager to continue writing the response even after it has received a FIN from the client.

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

setProxyCloseByLocation( location, 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 the traffic manager is responding to the request itself, enabling this setting will cause the traffic manager to continue writing the response even after it has received a FIN from the client. This is a location specific function, any action will operate on the specified location.

```java
void setProxyCloseByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set expect connections to the traffic manager to be prefixed with a PROXY protocol header.

```java
void setProxyProtocol(
    String[] names
    VirtualServer.Protocol[] values
)
```
Function Reference

VirtualServer

void setProxyProtocol(
    String[] names
    Boolean[] values
)

setProxyProtocolByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set expect connections to the traffic manager to be prefixed with a PROXY protocol header. This is a location specific function, any action will operate on the specified location.

void setProxyProtocolByLocation(
    String location
    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
)

setRTSPPortRangeByLocation( location, names, range ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the port range used for RTSP streaming data connections for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

void setRTSPPortRangeByLocation(
    String location
    String[] names
    VirtualServer.PortRange[] range
)

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

Set the time, in seconds, after which data-streams associated with RTSP connections timeout if no data is transmitted.

void setRTSPStreamingTimeout(
    String[] names
    Unsigned Integer[] values
)

setRTSPStreamingTimeoutByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time, in seconds, after which data-streams associated with RTSP connections timeout if no data is transmitted. This is a location specific function, any action will operate on the specified location.

void setRTSPStreamingTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
void setRecentConnsEnabled(String[] names, Boolean[] values)

void setRecentConnsEnabledByLocation(String location, String[] names, Boolean[] values)

void setRecentConnsSaveAll(String[] names, Boolean[] values)

void setRecentConnsSaveAllByLocation(String location, String[] names, Boolean[] values)

void setRequestSyslogEnabled(String[] names, Boolean[] values)
**setRequestSyslogEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether each of the named virtual servers should log each connection to a remote syslog server. This is a location specific function, any action will operate on the specified location.

```java
void setRequestSyslogEnabledByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

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

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

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

**setRequestSyslogFormatByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the remote log line format for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setRequestSyslogFormatByLocation(
    String location,
    String[] names,
    String[] values
)
```

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

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

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

**setRequestSyslogIPEndpointByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the remote syslog endpoint for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setRequestSyslogIPEndpointByLocation(
    String location,
    String[] names,
    String[] values
)
```
setRequestSyslogMessageLenLimit( names, values ) throws InvalidInput,
ObjectDoesNotExist, DeploymentError
Set syslog message length limit.
void setRequestSyslogMessageLenLimit(
    String[] names
    Unsigned Integer[] values
)

setRequestSyslogMessageLenLimitByLocation( location, names, values ) throws
InvalidInput, ObjectDoesNotExist, DeploymentError
Set syslog message length limit. This is a location specific function, any action will operate on the specified
location.
void setRequestSyslogMessageLenLimitByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

setRequestTracingEnabled( names, values ) throws InvalidInput,
ObjectDoesNotExist, DeploymentError
Set whether to record a detailed list of processing history for each request.
void setRequestTracingEnabled( 
    String[] names
    Boolean[] values
)

setRequestTracingEnabledByLocation( location, names, values ) throws
InvalidInput, ObjectDoesNotExist, DeploymentError
Set whether to record a detailed list of processing history for each request. This is a location specific
function, any action will operate on the specified location.
void setRequestTracingEnabledByLocation( 
    String location
    String[] names
    Boolean[] values
)

setRequestTracingIO( names, values ) throws InvalidInput, ObjectDoesNotExist,
DeploymentError
Set whether to record a detailed list of every IO event in the processing history for each request.
void setRequestTracingIO(
    String[] names
    Boolean[] values
)

setRequestTracingIOByLocation( location, names, values ) throws InvalidInput,
ObjectDoesNotExist, DeploymentError
Set whether to record a detailed list of every IO event in the processing history for each request. This is a
location specific function, any action will operate on the specified location.
void setRequestTracingIOByLocation(
    String location
    String[] names
    Boolean[] 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
)

setResponseRulesByLocation( location, names, rules ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Set the rules that are run on server responses for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.
void setResponseRulesByLocation(
    String location
    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
)

setRewriteSIPURIByLocation( location, 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. This is a location specific function, any action will operate on the specified location.
void setRewriteSIPURIByLocation(
    String location
    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
)
VirtualServer

VirtualServer Function Reference

setRulesByLocation( location, names, rules ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the rules that are run on client requests for each of the named virtual servers. This is a location specific function, any action will operate on the specified location.

```java
void setRulesByLocation(
    String location
    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.

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

setSIPDangerousRequestModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSIPDangerousRequestModeByLocation(
    String location
    String[] names
    VirtualServer.SIPDangerousRequestMode[] values
)
```

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

Set whether to follow routing information in SIP requests.

```java
void setSIPFollowRoute(
    String[] names
    Boolean[] values
)
```

setSIPFollowRouteByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether to follow routing information in SIP requests. This is a location specific function, any action will operate on the specified location.

```java
void setSIPFollowRouteByLocation(
    String location
    String[] names
    Boolean[] values
)
```
Function Reference

VirtualServer

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

Set maximum memory per connection.

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

**setSIPMaxConnectionMemoryByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set maximum memory per connection. This is a location specific function, any action will operate on the specified location.

```java
void setSIPMaxConnectionMemoryByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

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

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

**setSIPModeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set which mode of operation the SIP virtual server should run in. This is a location specific function, any action will operate on the specified location.

```java
void setSIPModeByLocation(
    String location
    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.

```java
void setSIPPortRange(
    String[] names
    VirtualServer.PortRange[] range
)
**setSIPPortRangeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSIPPortRangeByLocation(
    String location,
    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.

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

**setSIPStreamingTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSIPStreamingTimeoutByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```

**setSIPTimeoutMessages( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set send a timed out response to the client and CANCEL request to the server when a transaction times out.

```java
void setSIPTimeoutMessages(
    String[] names,
    Boolean[] values
)
```

**setSIPTimeoutMessagesByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set send a timed out response to the client and CANCEL request to the server when a transaction times out. This is a location specific function, any action will operate on the specified location.

```java
void setSIPTimeoutMessagesByLocation(
    String location,
    String[] names,
    Boolean[] values
)
setSSLCertificate( names, certs ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

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
)

setSSLCertificateByLocation( location, names, certs ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

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. This is a location specific function, any action will operate on the specified location.

void setSSLCertificateByLocation(
    String location
    String[] names
    String[] certs
)

setSSLCertificates( names, certs ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the name of the SSL certificates that are 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 SSL Certificates before turning on SSL Decryption.

void setSSLCertificates(
    String[] names
    String[][] certs
)

setSSLCertificatesByLocation( location, names, certs ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the name of the SSL certificates that are 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 SSL Certificates before turning on SSL Decryption. This is a location specific function, any action will operate on the specified location.

void setSSLCertificatesByLocation(
    String location
    String[] names
    String[][] certs
)

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

Set the ciphers allowed for connections to this virtual server

void setSSLCiphers(
    String[] names
)
String[] values
}

void setSSLCiphersByLocation(
    String location
    String[] names
    String[] values
)

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

void setSSLClientCertificateAuthoritiesByLocation(
    String location
    String[] names
    String[][] values
)

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

void setSSLClientCertificateHeadersByLocation(
    String location
    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.

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

setSSLDecryptByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSSLDecryptByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set the elliptic curve preference list for SSL connections to this virtual server.

```java
void setSSLEllipticCurves(
    String[] names
    String[] values
)
```

setSSLEllipticCurvesByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the elliptic curve preference list for SSL connections to this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setSSLEllipticCurvesByLocation(
    String location
    String[] names
    String[] 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.

```java
void setSSLExpectStartTLS(
    String[] names
    Boolean[] values
)
```
setSSLExpectStartTLSByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should upgrade SMTP connections to SSL using the STARTTLS command. This is a location specific function, any action will operate on the specified location.

```java
void setSSLExpectStartTLSByLocation(
    String location
    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.

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

setSSLHeadersByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSSLHeadersByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set whether Fallback SCSV is honored by this virtual server

```java
void setSSLHonorFallbackSCSV(
    String[] names
    VirtualServer.SSLHonorFallbackSCSV[] values
)
```

setSSLHonorFallbackSCSVByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether Fallback SCSV is honored by this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setSSLHonorFallbackSCSVByLocation(
    String location
    String[] names
    VirtualServer.SSLHonorFallbackSCSV[] 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.getLogConnectionFailures to control this configuration.

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

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

Set the CAs for which any client certificate they validate is considered valid even if the client certificate's expiration date has passed.

```java
void setSSLNeverExpiringClientCertificateAuthorities(
    String[] names
    String[][] values
)
```

setSSLNeverExpiringClientCertificateAuthoritiesByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the CAs for which any client certificate they validate is considered valid even if the client certificate's expiration date has passed. This is a location specific function, any action will operate on the specified location.

```java
void setSSLNeverExpiringClientCertificateAuthoritiesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Set the number of certificates in a certificate chain beyond those listed as NeverExpiringClientCertificateAuthorities whose certificate expiry will not be checked.

```java
void setSSLNeverExpiringClientCertificateAuthoritiesDepth(
    String[] names
    Unsigned Integer[] values
)
```

setSSLNeverExpiringClientCertificateAuthoritiesDepthByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of certificates in a certificate chain beyond those listed as NeverExpiringClientCertificateAuthorities whose certificate expiry will not be checked. This is a location specific function, any action will operate on the specified location.

```java
void setSSLNeverExpiringClientCertificateAuthoritiesDepthByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```
setSSLOCSPDefaults( names, ssl_ocsp_issuers ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the default OCSP responder settings for all client certificates.

```java
void setSSLOCSPDefaults(
    String[] names
    VirtualServer.SSLOCSPIssuer[] ssl_ocsp_issuers
)
```

setSSLOCSPDefaultsByLocation( location, names, ssl_ocsp_issuers ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the default OCSP responder settings for all client certificates. This is a location specific function, any action will operate on the specified location.

```java
void setSSLOCSPDefaultsByLocation(
    String location
    String[] names
    VirtualServer.SSLOCSPIssuer[] ssl_ocsp_issuers
)
```

setSSLOCSPIssuers( names, ssl_ocsp_issuers ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets a list of mappings between Certificate Authorities and OCSP responder settings. Certificates issued by these authorities will be verified with OCSP using these settings.

```java
void setSSLOCSPIssuers(
    String[] names
    VirtualServer.SSLOCSPIssuer[][] ssl_ocsp_issuers
)
```

setSSLOCSPIssuersByLocation( location, names, ssl_ocsp_issuers ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets a list of mappings between Certificate Authorities and OCSP responder settings. Certificates issued by these authorities will be verified with OCSP using these settings. This is a location specific function, any action will operate on the specified location.

```java
void setSSLOCSPIssuersByLocation(
    String location
    String[] names
    VirtualServer.SSLOCSPIssuer[][] ssl_ocsp_issuers
)
```

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

Set the number of seconds for which an OCSP response is considered valid if it has not yet exceeded the time specified in the 'nextUpdate' field.

```java
void setSSLOCSPMaxResponseAge(
    String[] names
    Unsigned Integer[] values
)
```
Function Reference

VirtualServer

`setSSLOCSPPMaxResponseAgeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError`

Set the number of seconds for which an OCSP response is considered valid if it has not yet exceeded the time specified in the 'nextUpdate' field. This is a location specific function, any action will operate on the specified location.

```java
void setSSLOCSPPMaxResponseAgeByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```

`setSSLOCSPPStapling( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError`

Set whether the traffic manager is allowed to provide OCSP responses for certificates as part of the handshake, if the client sends a TLS status_request extension in the ClientHello, and OCSP URIs are present in certificates used by this virtual server.

```java
void setSSLOCSPPStapling(
    String[] names,
    Boolean[] values
)
```

`setSSLOCSPPStaplingByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError`

Set whether the traffic manager is allowed to provide OCSP responses for certificates as part of the handshake, if the client sends a TLS status_request extension in the ClientHello, and OCSP URIs are present in certificates used by this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setSSLOCSPPStaplingByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

`setSSLOCSPTimeTolerance( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError`

Set the number of seconds outside the permitted range for which the 'thisUpdate' and 'nextUpdate' fields of an OCSP response are still considered valid.

```java
void setSSLOCSPTimeTolerance(
    String[] names,
    Unsigned Integer[] values
)
```

`setSSLOCSPTimeToleranceByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError`

Set the number of seconds outside the permitted range for which the 'thisUpdate' and 'nextUpdate' fields of an OCSP response are still considered valid. This is a location specific function, any action will operate on the specified location.

```java
void setSSLOCSPTimeToleranceByLocation(
    String location
)
setSSLOCSPTTimeout( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of seconds after which OCSP requests will be timed out

void setSSLOCSPTTimeout(
    String[] names
    Unsigned Integer[] values
)

setSSLOCSPTTimeoutByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the number of seconds after which OCSP requests will be timed out. This is a location specific function, any action will operate on the specified location.

void setSSLOCSPTTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

setSSLPreferSSLv3( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

This method is now deprecated.

void setSSLPreferSSLv3(
    String[] names
    Boolean[] values
)

setSSLPreferSSLv3ByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

This method is now deprecated. This is a location specific function, any action will operate on the specified location.

void setSSLPreferSSLv3ByLocation(
    String location
    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
)
Function Reference

setSSLRequestClientCertModeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should request (or require) an identifying certificate from each client. This is a location specific function, any action will operate on the specified location.

void setSSLRequestClientCertModeByLocation(
    String location
    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
)

setSSLSendCloseAlertsByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should send a close alert when initiating SSL socket disconnections. This is a location specific function, any action will operate on the specified location.

void setSSLSendCloseAlertsByLocation(
    String location
    String[] names
    Boolean[] values
)

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

Set the SSL signature algorithms preference list for SSL connections to this virtual server

void setSSLSignatureAlgorithms(
    String[] names
    String[] values
)

setSSLSignatureAlgorithmsByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the SSL signature algorithms preference list for SSL connections to this virtual server This is a location specific function, any action will operate on the specified location.

void setSSLSignatureAlgorithmsByLocation(
    String location
    String[] names
    String[] values
)
setSSLSupportSSL2( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

This method is now deprecated.

void setSSLSupportSSL2(
    String[] names
    VirtualServer.SSLSupportSSL2[] values
)

setSSLSupportSSL2ByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

This method is now deprecated. This is a location specific function, any action will operate on the specified location.

void setSSLSupportSSL2ByLocation(
    String location
    String[] names
    VirtualServer.SSLSupportSSL2[] values
)

setSSLSupportSSL3( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether SSLv3 is enabled for this virtual server

void setSSLSupportSSL3(
    String[] names
    VirtualServer.SSLSupportSSL3[] values
)

setSSLSupportSSL3ByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether SSLv3 is enabled for this virtual server. This is a location specific function, any action will operate on the specified location.

void setSSLSupportSSL3ByLocation(
    String location
    String[] names
    VirtualServer.SSLSupportSSL3[] values
)

setSSLSupportTLS1( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether TLSv1.0 is enabled for this virtual server

void setSSLSupportTLS1(
    String[] names
    VirtualServer.SSLSupportTLS1[] values
)

setSSLSupportTLS11( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether TLSv1.1 is enabled for this virtual server

void setSSLSupportTLS11(
setSSLSupportTLS11ByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether TLSv1.1 is enabled for this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportTLS11ByLocation(
    String location
    String[] names
    VirtualServer.SSLSupportTLS11[] values
)
```

setSSLSupportTLS12( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether TLSv1.2 is enabled for this virtual server.

```java
void setSSLSupportTLS12(
    String[] names
    VirtualServer.SSLSupportTLS12[] values
)
```

setSSLSupportTLS12ByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether TLSv1.2 is enabled for this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportTLS12ByLocation(
    String location
    String[] names
    VirtualServer.SSLSupportTLS12[] values
)
```

setSSLSupportTLS1ByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether TLSv1.0 is enabled for this virtual server. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportTLS1ByLocation(
    String location
    String[] names
    VirtualServer.SSLSupportTLS1[] 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.

```java
void setSSLTrustMagic(
    String[] names
    Boolean[] values
)
setSSLTrustMagicByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSSLTrustMagicByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

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

Set whether or not the traffic manager should use OCSP to check the revocation status of client certificates

```java
void setSSLUseOCSP(
    String[] names,
    Boolean[] values
)
```

setSSLUseOCSPByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether or not the traffic manager should use OCSP to check the revocation status of client certificates. This is a location specific function, any action will operate on the specified location.

```java
void setSSLUseOCSPByLocation(
    String location,
    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.

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

setServerfirstBannerByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setServerfirstBannerByLocation(
    String location,
    String[] names,
    String[] values
)
```
Function Reference

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

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

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

**setServiceLevelMonitoringByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set the Service Level Monitoring class that each of the named virtual servers uses. This is a location specific function, any action will operate on the specified location.

```java
void setServiceLevelMonitoringByLocation(
    String location
    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.

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

**setSipTransactionTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSipTransactionTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setStripXForwardedProtoHeader( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set Whether or not the virtual server should strip the \ 'X-Forwarded-Proto' header from incoming requests.

```java
void setStripXForwardedProtoHeader(
    String[] names
    Boolean[] values
)
```
**setStripXForwardedProtoHeaderByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set Whether or not the virtual server should strip the \`X-Forwarded-Proto\' header from incoming requests. This is a location specific function, any action will operate on the specified location.

```java
void setStripXForwardedProtoHeaderByLocation(
    String location
    String[] names
    Boolean[] 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.

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

**setTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setTransactionExportBrief( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether to export a restricted set of metadata about transactions processed by this virtual server. If enabled, more verbose information such as client and server headers and request tracing events will be omitted from the exported data.

```java
void setTransactionExportBrief(
    String[] names
    Boolean[] values
)
```

**setTransactionExportBriefByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether to export a restricted set of metadata about transactions processed by this virtual server. If enabled, more verbose information such as client and server headers and request tracing events will be omitted from the exported data. This is a location specific function, any action will operate on the specified location.

```java
void setTransactionExportBriefByLocation(
    String location
    String[] names
    Boolean[] values
)
Function Reference

VirtualServer

```java

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

Set whether to export metadata about transactions handled by this service to the globally configured endpoint.

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

```java

setTransactionExportEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether to export metadata about transactions handled by this service to the globally configured endpoint. This is a location specific function, any action will operate on the specified location.

```void setTransactionExportEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)

```java

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

Set the set of HTTP header names for which corresponding values should be redacted from the metadata exported by this virtual server from transaction export.

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

```java

setTransactionExportHTTPHeaderBlacklistByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the set of HTTP header names for which corresponding values should be redacted from the metadata exported by this virtual server from transaction export. This is a location specific function, any action will operate on the specified location.

```void setTransactionExportHTTPHeaderBlacklistByLocation(
    String location
    String[] names
    String[][] values
)

```java

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

Set whether the transaction processing timeline included in the metadata export is recorded with a high, microsecond, resolution. If set to No, timestamps will be recorded with a resolution of milliseconds.

```void setTransactionExportHiRes(
    String[] names
    Boolean[] values
)```
setTransactionExportHiResByLocation( location, names, values ) throws
InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether the transaction processing timeline included in the metadata export is recorded with a high,
microsecond, resolution. If set to No, timestamps will be recorded with a resolution of milliseconds. This is
a location specific function, any action will operate on the specified location.

void setTransactionExportHiResByLocation(
    String location
    String[] names
    Boolean[] values
)

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

Set whether or not bound sockets should be configured for transparent proxying.

void setTransparent(
    String[] names
    Boolean[] values
)

setTransparentByLocation( location, names, values ) throws InvalidInput,
ObjectDoesNotExist, DeploymentError

Set whether or not bound sockets should be configured for transparent proxying. This is a location specific
function, any action will operate on the specified location.

void setTransparentByLocation(
    String location
    String[] names
    Boolean[] values
)

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

Set when the traffic manager should consider a UDP transaction to have ended.

void setUDPEndTransaction(
    String[] names
    VirtualServer.UDPEndTransaction[] values
)

setUDPEndTransactionByLocation( location, names, values ) throws InvalidInput,
ObjectDoesNotExist, DeploymentError

Set when the traffic manager should consider a UDP transaction to have ended. This is a location specific
function, any action will operate on the specified location.

void setUDPEndTransactionByLocation(
    String location
    String[] names
    VirtualServer.UDPEndTransaction[] values
)
**setUDPEndpointPersistence( names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether each of the named virtual servers should send UDP datagrams received from the same IP address and port to the same pool node if they match an existing UDP session. Sessions are defined by the protocol being handled, for example SIP datagrams are grouped based on the value of the Call-ID header.

```java
void setUDPEndpointPersistence(
    String[] names
    Boolean[] values
)
```

**setUDPEndpointPersistenceByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError**

Set whether each of the named virtual servers should send UDP datagrams received from the same IP address and port to the same pool node if they match an existing UDP session. Sessions are defined by the protocol being handled, for example SIP datagrams are grouped based on the value of the Call-ID header. This is a location specific function, any action will operate on the specified location.

```java
void setUDPEndpointPersistenceByLocation(
    String location
    String[] names
    Boolean[] 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.

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

**setUDPResponseDatagramsExpectedByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setUDPResponseDatagramsExpectedByLocation(
    String location
    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.

```java
void setUDPTimeout(
```
VirtualServer

```java
String[] names
Unsigned Integer[] values
}

setUDPTimeoutByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void setUDPTimeoutByLocation(
    String location
    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
}

setUseNagleByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether Nagle's algorithm should be used for TCP connections. This is a location specific function, any action will operate on the specified location.

void setUseNagleByLocation(
    String location
    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
}

setWebcacheControlOutByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the Cache-Control header that should be sent with cached HTTP responses. This is a location specific function, any action will operate on the specified location.

void setWebcacheControlOutByLocation(
    String location
    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.

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

setWebcacheEnabledByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set whether each of the named virtual servers should attempt to cache web server responses. This is a location specific function, any action will operate on the specified location.

```java
void setWebcacheEnabledByLocation(
    String location,
    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.

```java
void setWebcacheErrorpageTime(
    String[] names,
    Unsigned Integer[] values
)
```

setWebcacheErrorpageTimeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time periods that each of the named virtual servers should cache error pages for. This is a location specific function, any action will operate on the specified location.

```java
void setWebcacheErrorpageTimeByLocation(
    String location,
    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.

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

setWebcacheRefreshTimeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time periods that each of the named virtual servers should consider re-fetching cached pages in. This is a location specific function, any action will operate on the specified location.
VirtualServer

void setWebcacheRefreshTimeByLocation{
   String location
   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
}

setWebcacheTimeByLocation( location, names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the time periods that each of the named virtual servers should cache web pages for. This is a location specific function, any action will operate on the specified location.

void setWebcacheTimeByLocation{
   String location
   String[] names
   Unsigned Integer[] values
}

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.

```plaintext
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.

```plaintext
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.

```plaintext
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.SSLOCSPIssuer**

This object represents a mapping between a Certificate Authority (this is the name of an item in the Certificate Authorities Catalog) and configuration for an OCSP responder. Certificates issued by the Certificate Authority will use these OCSP responder settings.

```plaintext
struct VirtualServer.SSLOCSPIssuer {
    # The Certificate Authority for which these settings apply. This is the name of an item in the Certificate Authorities Catalog.
    String ca;

    # The URL of the OCSP responder that should be used to check the revocation status of certificates issued by the Certificate Authority.
    String url;

    # Is OCSP required for certificates signed by this CA?
    VirtualServer.SSLOCSPCheck required;

    # If set to true the Authority Information Access X509 extension will be used to determine the OCSP server's URL
    Boolean aia;

    # Should an OCSP nonce be added to each request to protect against replay
    String nonce;
}
```
# attacks. Not all OCSP servers support nonces.
VirtualServer.SSLOCSPNonce nonce;

# Should we sign OCSP requests?
VirtualServer.SSLOCSPSignMode sign_mode;

# The key pair used to sign OCSP requests. If not set OCSP requests will not
# be signed. Must be an entry in the SSL Certificates Catalog.
String signer;

# The expected certificate that the OCSP responder should provide. Must be in
# the Certificate Authority catalog, or be empty (meaning the issuer
# certificate), or be exactly "_SIGNED_BY_ISSUER_" (which will accept either
# the issuer or one that is signed by it and has id-kp-OCSPSigning in
# extendedKeyUsage and has id-pkix-ocsp-nocheck).
String responder_cert;
}

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;
}

VirtualServer.SSLSiteAlt

This object is an extension to SSLSite and provides support for multiple certificates per SSL site.

struct VirtualServer.SSLSiteAlt  {
    # 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;

    # The extra SSL certificates.
    String[] alt_certificates;
}

Enumerations

VirtualServer.CompressionETagRewrite

enum VirtualServer.CompressionETagRewrite ( 
    # Leave the ETag unchanged
    ignore,

    # Delete the ETag header
    delete,

    # Change the ETag header to specify a weak match

weaken,

  # Wrap the ETag, and attempt to unwrap safe conditional requests
  wrap
}

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.DNSRecordsetOrder

enum VirtualServer.DNSRecordsetOrder {
  # Fixed
  fixed,

  # Cyclic
  cyclic
}

VirtualServer.HttpChunkOverheadForwarding

enum VirtualServer.HttpChunkOverheadForwarding {
  # lazy
  lazy,

  # eager
  eager
}

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**

```plaintext
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)
```
VirtualServer.

VirtualServer.RuleRunFlag

This enumeration defines the run flags for a particular rule.

def VirtualServer.RuleRunFlag {
    # Run on every request or response.
    run_every,
    # Run only on the first request or response.
    only_first
}

VirtualServer.SIPDangerousRequestMode

def 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

def VirtualServer.SIPMode {
    # SIP Routing
    route,
# SIP Gateway
```
sipgw,
```
# Full Gateway
```
fullgw
```
# Use the nonce extension in requests. The response must contain the correct nonce, otherwise it is rejected.

```
strict
```

**VirtualServer.SSLOCSPSignMode**

The different modes for OCSP request signing

```
enum VirtualServer.SSLOCSPSignMode {
    # Do not sign OCSP requests
    none,

    # Use default OCSP settings for signing requests
    use_default,

    # Use a specific catalog certificate to sign requests
    sign
}
```

**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
}
```

**VirtualServer.SSLSupportSSL2**

```
enum VirtualServer.SSLSupportSSL2 {
    # Use the global setting for SSLv2
    use_default,

    # Enable SSLv2 (not recommended)
    enabled,

    # Disable SSLv2
    disabled
}
```

**VirtualServer.SSLSupportSSL3**

```
enum VirtualServer.SSLSupportSSL3 {
    # Use the global setting for SSLv3
    use_default,

    # Enable SSLv3
    enabled,

    # Disable SSLv3
    disabled
}
```
VirtualServer.SSLSupportTLS1

```java
enum VirtualServer.SSLSupportTLS1 {
    # Use the global setting for TLSv1.0
    use_default,
    # Enable TLSv1.0
    enabled,
    # Disable TLSv1.0
    disabled
}
```

VirtualServer.SSLSupportTLS11

```java
enum VirtualServer.SSLSupportTLS11 {
    # Use the global setting for TLSv1.1
    use_default,
    # Enable TLSv1.1
    enabled,
    # Disable TLSv1.1
    disabled
}
```

VirtualServer.SSLSupportTLS12

```java
enum VirtualServer.SSLSupportTLS12 {
    # Use the global setting for TLSv1.2
    use_default,
    # Enable TLSv1.2
    enabled,
    # Disable TLSv1.2
    disabled
}
```

VirtualServer.UDPEndTransaction

```java
enum VirtualServer.UDPEndTransaction {
    # When they time out
    timeout,
    # After one response
    one_response,
    # When the number of responses matches the number of requests
    match_requests
}
```

---

**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.
## Methods

### addAutoScaledPool( names, nodes ) throws ObjectAlreadyExists, InvalidObjectName, InvalidInput, DeploymentError

Add each of the named autoscaled pools, using the node lists for each. The node lists can be empty.

```java
void addAutoScaledPool(
    String[] names
    String[][] nodes
)
```

### addAutoscaleSecuritygroupids( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add the security group IDs to associate to the new EC2 instances.

```java
void addAutoscaleSecuritygroupids(
    String[] names
    String[][] values
)
```

### addAutoscaleSecuritygroupidsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add the security group IDs to associate to the new EC2 instances. This is a location specific function, any action will operate on the specified location.

```java
void addAutoscaleSecuritygroupidsByLocation(
    String location
    String[] names
    String[][] values
)
```

### addAutoscaleSubnetids( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic.

```java
void addAutoscaleSubnetids(
    String[] names
    String[][] values
)
```

### addAutoscaleSubnetidsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic. This is a location specific function, any action will operate on the specified location.

```java
void addAutoscaleSubnetidsByLocation(
    String location
    String[] names
    String[][] values
)
```
addDNSAutoscaleHostnames( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add the hostnames to be used for DNS-derived autoscaling

```java
void addDNSAutoscaleHostnames(
    String[] names
    String[][] values
)
```

addDNSAutoscaleHostnamesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add the hostnames to be used for DNS-derived autoscaling This is a location specific function, any action will operate on the specified location.

```java
void addDNSAutoscaleHostnamesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

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

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

addDrainingNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Add nodes to the lists of draining nodes, for each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void addDrainingNodesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Add monitors to each of the named pools.

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

addMonitorsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

Add monitors to each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void addMonitorsByLocation(
    String location
    String[] names
    String[][] values
)
Function Reference

**Pool**

```java
void addMonitorsByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Add nodes to each of the named pools.

```java
void addNodes{
    String[] names
    String[][] values
}
```

**addNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Add nodes to each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void addNodesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

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

```java
void addPool{
    String[] names
    String[][] nodes
}
```

**addSSLCommonNameMatch( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Add the list of names against which the 'common name' of the certificate is matched.

```java
void addSSLCommonNameMatch(
    String[] names
    String[][] values
)
```

**addSSLCommonNameMatchByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Add the list of names against which the 'common name' of the certificate is matched. This is a location specific function, any action will operate on the specified location.

```java
void addSSLCommonNameMatchByLocation(
    String location
    String[] names
    String[][] values
)
```
copyPool( names, new_names ) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError

Copy each of the named pools.

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

deletePool( names ) throws ObjectInUse, ObjectDoesNotExist, DeploymentError

Delete each of the named pools.

```java
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.

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

disableNodesByLocation( location, names, nodes ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

For each of the named pools, disable the specified nodes in the pool. This is a location specific function, any action will operate on the specified location.

```java
void disableNodesByLocation(
    String location
    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.

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

enableNodesByLocation( location, names, nodes ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

For each of the named pools, enable the specified nodes that are disabled in the pool. This is a location specific function, any action will operate on the specified location.

```java
void enableNodesByLocation(
    String location
    String[] names
    String[][] nodes
)
getAutoscaleAddnodeDelaytime( names ) throws ObjectDoesNotExist
Get Delay in seconds before the node should be added to the autoscaled pool
Unsigned Integer[] getAutoscaleAddnodeDelaytime(
  String[] names
)

getAutoscaleAddnodeDelaytimeByLocation( location, names ) throws ObjectDoesNotExist
Get Delay in seconds before the node should be added to the autoscaled pool This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getAutoscaleAddnodeDelaytimeByLocation(
  String location
  String[] names
)

getAutoscaleCloudcredentials( names ) throws ObjectDoesNotExist
Get the cloud credentials for this autoscaled pool
String[] getAutoscaleCloudcredentials(
  String[] names
)

getAutoscaleCloudcredentialsByLocation( location, names ) throws ObjectDoesNotExist
Get the cloud credentials for this autoscaled pool This is a location specific function, any action will operate on the specified location.
String[] getAutoscaleCloudcredentialsByLocation(
  String location
  String[] names
)

getAutoscaleCluster( names ) throws ObjectDoesNotExist
Get The ESX host or ESX cluster name to put the new virtual machine instances on.
String[] getAutoscaleCluster(
  String[] names
)

getAutoscaleClusterByLocation( location, names ) throws ObjectDoesNotExist
Get The ESX host or ESX cluster name to put the new virtual machine instances on. This is a location specific function, any action will operate on the specified location.
String[] getAutoscaleClusterByLocation(
  String location
  String[] names
)
getAutoscaleDatacenter( names ) throws ObjectDoesNotExist

Get The name of the logical datacenter on the vCenter server.

```java
String[] getAutoscaleDatacenter(
    String[] names
)
```

getAutoscaleDatacenterByLocation( location, names ) throws ObjectDoesNotExist

Get The name of the logical datacenter on the vCenter server. This is a location specific function, any action will operate on the specified location.

```java
String[] getAutoscaleDatacenterByLocation(
    String location
    String[] names
)
```

getAutoscaleDatastore( names ) throws ObjectDoesNotExist

Get The name of the datastore to be used by the newly created virtual machine.

```java
String[] getAutoscaleDatastore(
    String[] names
)
```

getAutoscaleDatastoreByLocation( location, names ) throws ObjectDoesNotExist

Get The name of the datastore to be used by the newly created virtual machine. This is a location specific function, any action will operate on the specified location.

```java
String[] getAutoscaleDatastoreByLocation(
    String location
    String[] names
)
```

getAutoscaleEnabled( names ) throws ObjectDoesNotExist

Get whether this pool uses autoscaling.

```java
Boolean[] getAutoscaleEnabled(
    String[] names
)
```

getAutoscaleEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether this pool uses autoscaling. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getAutoscaleEnabledByLocation(
    String location
    String[] names
)
```

getAutoscaleExternal( names ) throws ObjectDoesNotExist

Get whether autoscaling is handled externally or internally.

```java
Boolean[] getAutoscaleExternal(
    String[] names
)
```
Function Reference Pool

getAutoscaleExternalByLocation( location, names ) throws ObjectDoesNotExist

Get whether autoscaling is handled externally or internally This is a location specific function, any action will operate on the specified location.

Boolean[] getAutoscaleExternalByLocation(
    String location
    String[] names
)

getAutoscaleExtraargs( names ) throws ObjectDoesNotExist

Get Any extra arguments to the autoscaling API. Each argument can be separated by comma. E.g in case of EC2, it can take extra parameters to the Amazon's RunInstance API say DisableApiTermination=false,Placement.Tenancy=default.

String[] getAutoscaleExtraargs(
    String[] names
)

getAutoscaleExtraargsByLocation( location, names ) throws ObjectDoesNotExist

Get Any extra arguments to the autoscaling API. Each argument can be separated by comma. E.g in case of EC2, it can take extra parameters to the Amazon's RunInstance API say DisableApiTermination=false,Placement.Tenancy=default. This is a location specific function, any action will operate on the specified location.

String[] getAutoscaleExtraargsByLocation(
    String location
    String[] names
)

getAutoscaleHysteresis( names ) throws ObjectDoesNotExist

Get the hysteresis period for an autoscaled pool

Unsigned Integer[] getAutoscaleHysteresis(
    String[] names
)

getAutoscaleHysteresisByLocation( location, names ) throws ObjectDoesNotExist

Get the hysteresis period for an autoscaled pool This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleHysteresisByLocation(
    String location
    String[] names
)

getAutoscaleImageid( names ) throws ObjectDoesNotExist

Get the image identifier

String[] getAutoscaleImageid(
    String[] names
)
Pool

Function Reference

getAutoscaleImageidByLocation( location, names ) throws ObjectDoesNotExist

Get the image identifier. This is a location specific function, any action will operate on the specified location.

String[] getAutoscaleImageidByLocation(
    String location
    String[] names
)

getAutoscaleIpstouse( names ) throws ObjectDoesNotExist

Get whether to use the public or private IPs

Pool.AutoscaleIpstouse[] getAutoscaleIpstouse(
    String[] names
)

getAutoscaleIpstouseByLocation( location, names ) throws ObjectDoesNotExist

Get whether to use the public or private IPs. This is a location specific function, any action will operate on the specified location.

Pool.AutoscaleIpstouse[] getAutoscaleIpstouseByLocation(
    String location
    String[] names
)

getAutoscaleLastnodeIdletime( names ) throws ObjectDoesNotExist

Get the idle time of the last node in an autoscaled pool before it can be destroyed

Unsigned Integer[] getAutoscaleLastnodeIdletime(
    String[] names
)

getAutoscaleLastnodeIdletimeByLocation( location, names ) throws ObjectDoesNotExist

Get the idle time of the last node in an autoscaled pool before it can be destroyed. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleLastnodeIdletimeByLocation(
    String location
    String[] names
)

getAutoscaleMaxNodes( names ) throws ObjectDoesNotExist

Get the maximum number of nodes in an autoscaled pool

Unsigned Integer[] getAutoscaleMaxNodes(
    String[] names
)

getAutoscaleMaxNodesByLocation( location, names ) throws ObjectDoesNotExist

Get the maximum number of nodes in an autoscaled pool. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleMaxNodesByLocation(
    String location
)
String[] names

getAutoscaleMinNodes( names ) throws ObjectDoesNotExist
Get the minimum number of nodes in an autoscaled pool
Unsigned Integer[] getAutoscaleMinNodes(
    String[] names
)

getAutoscaleMinNodesByLocation( location, names ) throws ObjectDoesNotExist
Get the minimum number of nodes in an autoscaled pool. This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getAutoscaleMinNodesByLocation(
    String location
    String[] names
)

getAutoscaleName( names ) throws ObjectDoesNotExist
Get the node name prefix for this autoscaled pool
String[] getAutoscaleName(
    String[] names
)

getAutoscaleNameByLocation( location, names ) throws ObjectDoesNotExist
Get the node name prefix for this autoscaled pool. This is a location specific function, any action will operate on the specified location.
String[] getAutoscaleNameByLocation(
    String location
    String[] names
)

getAutoscalePort( names ) throws ObjectDoesNotExist
Get the port number for this autoscaled pool
Unsigned Integer[] getAutoscalePort(
    String[] names
)

getAutoscalePortByLocation( location, names ) throws ObjectDoesNotExist
Get the port number for this autoscaled pool. This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getAutoscalePortByLocation(
    String location
    String[] names
)

getAutoscaleRefractory( names ) throws ObjectDoesNotExist
Get the refractory period for an autoscaled pool
Unsigned Integer[] getAutoscaleRefractory(String[] names)

getAutoscaleRefractoryByLocation(location, names) throws ObjectDoesNotExist

Get the refractory period for an autoscaled pool. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleRefractoryByLocation(String location, String[] names)

getAutoscaleResponseTime(names) throws ObjectDoesNotExist

Get the expected node response time in milliseconds.

Unsigned Integer[] getAutoscaleResponseTime(String[] names)

getAutoscaleResponseTimeByLocation(location, names) throws ObjectDoesNotExist

Get the expected node response time in milliseconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleResponseTimeByLocation(String location, String[] names)

getAutoscaleScaledownLevel(names) throws ObjectDoesNotExist

Get the threshold of conforming requests for scaling down.

Unsigned Integer[] getAutoscaleScaledownLevel(String[] names)

getAutoscaleScaledownLevelByLocation(location, names) throws ObjectDoesNotExist

Get the threshold of conforming requests for scaling down. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleScaledownLevelByLocation(String location, String[] names)

getAutoscaleScaleupLevel(names) throws ObjectDoesNotExist

Get the acceptable lower percentage of conforming requests.

Unsigned Integer[] getAutoscaleScaleupLevel(String[] names)
getAutoscaleScaleupLevelByLocation( location, names ) throws ObjectDoesNotExist

Get the acceptable lower percentage of conforming requests. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getAutoscaleScaleupLevelByLocation(
    String location
    String[] names
)

getAutoscaleSecuritygroupids( names ) throws ObjectDoesNotExist

Get the security group IDs to associate to the new EC2 instances.

String[][] getAutoscaleSecuritygroupids(
    String[] names
)

getAutoscaleSecuritygroupidsByLocation( location, names ) throws ObjectDoesNotExist

Get the security group IDs to associate to the new EC2 instances. This is a location specific function, any action will operate on the specified location.

String[][] getAutoscaleSecuritygroupidsByLocation(
    String location
    String[] names
)

getAutoscaleSizeid( names ) throws ObjectDoesNotExist

Get the size identifier.

String[] getAutoscaleSizeid(
    String[] names
)

getAutoscaleSizeidByLocation( location, names ) throws ObjectDoesNotExist

Get the size identifier. This is a location specific function, any action will operate on the specified location.

String[] getAutoscaleSizeidByLocation(
    String location
    String[] names
)

getAutoscaleSubnetids( names ) throws ObjectDoesNotExist

Get the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic.

String[][] getAutoscaleSubnetids(
    String[] names
)
getAutoscaleSubnetidsByLocation( location, names ) throws ObjectDoesNotExist

Get the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic. This is a location specific function, any action will operate on the specified location.

```java
String[][] getAutoscaleSubnetidsByLocation(
        String location
        String[] names
    )
```

getBandwidthClass( names ) throws ObjectDoesNotExist

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

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

getBandwidthClassByLocation( location, names ) throws ObjectDoesNotExist

Get the Bandwidth Classes that each of the named pools uses. This is a location specific function, any action will operate on the specified location.

```java
String[] getBandwidthClassByLocation(
        String location
        String[] names
    )
```

getDNSAutoscaleEnabled( names ) throws ObjectDoesNotExist

Get whether this pool uses DNS-derived autoscaling

```java
Boolean[] getDNSAutoscaleEnabled(
        String[] names
    )
```

getDNSAutoscaleEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether this pool uses DNS-derived autoscaling. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getDNSAutoscaleEnabledByLocation(
        String location
        String[] names
    )
```

getDNSAutoscaleHostnames( names ) throws ObjectDoesNotExist

Get the hostnames to be used for DNS-derived autoscaling

```java
String[][] getDNSAutoscaleHostnames(
        String[] names
    )
```
getDNSAutoscaleHostnamesByLocation( location, names ) throws ObjectDoesNotExist

Get the hostnames to be used for DNS-derived autoscaling. This is a location specific function, any action will operate on the specified location.

```java
String[][] getDNSAutoscaleHostnamesByLocation(
    String location
    String[] names
)
```

getDNSAutoscalePort( names ) throws ObjectDoesNotExist

Get the port number for DNS-derived autoscaling in this pool

```java
Unsigned Integer[] getDNSAutoscalePort(
    String[] names
)
```

getDNSAutoscalePortByLocation( location, names ) throws ObjectDoesNotExist

Get the port number for DNS-derived autoscaling in this pool. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getDNSAutoscalePortByLocation(
    String location
    String[] names
)
```

getDisabledNodes( names ) throws ObjectDoesNotExist

For each of the named pools, get the disabled nodes in the pool.

```java
String[][] getDisabledNodes(
    String[] names
)
```

getDisabledNodesByLocation( location, names ) throws ObjectDoesNotExist

For each of the named pools, get the disabled nodes in the pool. This is a location specific function, any action will operate on the specified location.

```java
String[][] getDisabledNodesByLocation(
    String location
    String[] names
)
```

getDrainingNodes( names ) throws ObjectDoesNotExist

Get the lists of draining nodes for each of the named pools.

```java
String[][] getDrainingNodes(
    String[] names
)
```

getDrainingNodesByLocation( location, names ) throws ObjectDoesNotExist

Get the lists of draining nodes for each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
String[][] getDrainingNodesByLocation(
    String location
    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
)

getFTPSupportRfc2428ByLocation( location, names ) throws ObjectDoesNotExist

Get whether backend IPv4 nodes understand the FTP EPRT and EPSV commands. This is a location specific function, any action will operate on the specified location.

Boolean[] getFTPSupportRfc2428ByLocation(
    String location
    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
)

getFailpoolByLocation( location, names ) throws ObjectDoesNotExist

Get the pool to use when all nodes in a pool fail, for each of the named pools. This is a location specific function, any action will operate on the specified location.

String[] getFailpoolByLocation(
    String location
    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
)

getKeepaliveByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named pools should maintain HTTP keepalive connections to the nodes. This is a location specific function, any action will operate on the specified location.

Boolean[] getKeepaliveByLocation(
    String location
    String[] names
)
Function Reference Pool

Boolean[] getKeepaliveByLocation(
    String location
    String[] names
)

getKeepaliveNonIdempotent( names ) throws ObjectDoesNotExist
Get whether each of the named pools should maintain HTTP keepalive connections to the nodes for non-
idempotent requests.

Boolean[] getKeepaliveNonIdempotent(
    String[] names
)

getKeepaliveNonIdempotentByLocation( location, names ) throws
ObjectDoesNotExist
Get whether each of the named pools should maintain HTTP keepalive connections to the nodes for non-
idempotent requests. This is a location specific function, any action will operate on the specified location.

Boolean[] getKeepaliveNonIdempotentByLocation(
    String location
    String[] names
)

getKerberosProtocolTransitionPrincipal( names ) throws ObjectDoesNotExist
Get the Kerberos principal that each of the named pools uses to perform Kerberos Protocol Transition

String[] getKerberosProtocolTransitionPrincipal(
    String[] names
)

getKerberosProtocolTransitionPrincipalByLocation( location, names ) throws
ObjectDoesNotExist
Get the Kerberos principal that each of the named pools uses to perform Kerberos Protocol Transition. This
is a location specific function, any action will operate on the specified location.

String[] getKerberosProtocolTransitionPrincipalByLocation(
    String location
    String[] names
)

getKerberosProtocolTransitionTarget( names ) throws ObjectDoesNotExist
Get the Kerberos principal name of the service that each of the named pools target

String[] getKerberosProtocolTransitionTarget(
    String[] names
)

getKerberosProtocolTransitionTargetByLocation( location, names ) throws
ObjectDoesNotExist
Get the Kerberos principal name of the service that each of the named pools target. This is a location specific
function, any action will operate on the specified location.

String[] getKerberosProtocolTransitionTargetByLocation(
    String location
Pool

Function Reference

String[] names

getL4AccelSNAT( names ) throws ObjectDoesNotExist

Get whether connections to the back-end nodes should appear to originate from an IP address raised on the traffic manager, rather than the IP address from which they were received by the traffic manager.

Boolean[] getL4AccelSNAT(
    String[] names
)

getL4AccelSNATByLocation( location, names ) throws ObjectDoesNotExist

Get whether connections to the back-end nodes should appear to originate from an IP address raised on the traffic manager, rather than the IP address from which they were received by the traffic manager. This is a location specific function, any action will operate on the specified location.

Boolean[] getL4AccelSNATByLocation(
    String location
    String[] names
)

getLoadBalancingAlgorithm( names ) throws ObjectDoesNotExist

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

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

getLoadBalancingAlgorithmByLocation( location, names ) throws ObjectDoesNotExist

Get the load balancing algorithms that each of the named pools uses. This is a location specific function, any action will operate on the specified location.

Pool.LoadBalancingAlgorithm[] getLoadBalancingAlgorithmByLocation(
    String location
    String[] names
)

getMaxConnectTime( names ) throws ObjectDoesNotExist

Get the time 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
)

getMaxConnectTimeByLocation( location, names ) throws ObjectDoesNotExist

Get the time that each of the named pools should wait for a connection to establish to a node before trying another node, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxConnectTimeByLocation(
    String location
    String[] names
getMaxConnectionAttempts( names ) throws ObjectDoesNotExist

Get the number of times that each of the named pools can try to connect to any of its nodes before sending an error response.

Unsigned Integer[] getMaxConnectionAttempts(
    String[] names
)

getMaxConnectionAttemptsByLocation( location, names ) throws ObjectDoesNotExist

Get the number of times that each of the named pools can try to connect to any of its nodes before sending an error response. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxConnectionAttemptsByLocation(
    String location
    String[] names
)

getMaxConnectionsPernode( names ) throws ObjectDoesNotExist

Get is the maximum number of concurrent connections allowed to each node in the pool per machine.

Unsigned Integer[] getMaxConnectionsPernode(
    String[] names
)

getMaxConnectionsPernodeByLocation( location, names ) throws ObjectDoesNotExist

Get is the maximum number of concurrent connections allowed to each node in the pool per machine. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxConnectionsPernodeByLocation(
    String location
    String[] names
)

getMaxIdleConnectionsPerNode( names ) throws ObjectDoesNotExist

Get the maximum number of unused HTTP keepalive connections that each of the named pools should maintain to an individual node.

Unsigned Integer[] getMaxIdleConnectionsPerNode(
    String[] names
)

getMaxIdleConnectionsPerNodeByLocation( location, names ) throws ObjectDoesNotExist

Get the maximum number of unused HTTP keepalive connections that each of the named pools should maintain to an individual node. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxIdleConnectionsPerNodeByLocation(
    String location
    String[] names
)
getMaxKeepalivesPerNode( names ) throws ObjectDoesNotExist

getMaxKeepalivesPerNode is deprecated, please use getMaxIdleConnectionsPerNode instead.

Unsigned Integer[] getMaxKeepalivesPerNode(
    String[] names
)

getMaxKeepalivesPerNodeByLocation( location, names ) throws ObjectDoesNotExist

getMaxKeepalivesPerNode is deprecated, please use getMaxIdleConnectionsPerNode instead. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxKeepalivesPerNodeByLocation(
    String location
    String[] names
)

getMaxQueueSize( names ) throws ObjectDoesNotExist

Get is the maximum number of connections that can be queued due to connection limits.

Unsigned Integer[] getMaxQueueSize(
    String[] names
)

getMaxQueueSizeByLocation( location, names ) throws ObjectDoesNotExist

Get is the maximum number of connections that can be queued due to connection limits. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxQueueSizeByLocation(
    String location
    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
)

getMaxReplyTimeByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxReplyTimeByLocation(
    String location
    String[] names
)
Function Reference

**getMaxTimedOutConnectionAttempts( names ) throws ObjectDoesNotExist**

Get the number of times that each of the named pools can try to connect and time out waiting for a response, by exceeding max_reply_time, to any of its nodes before sending an error response.

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

**getMaxTimedOutConnectionAttemptsByLocation( location, names ) throws ObjectDoesNotExist**

Get the number of times that each of the named pools can try to connect and time out waiting for a response, by exceeding max_reply_time, to any of its nodes before sending an error response. This is a location specific function, any action will operate on the specified location.

```
Unsigned Integer[] getMaxTimedOutConnectionAttemptsByLocation(
    String location
    String[] names
)
```

**getMonitors( names ) throws ObjectDoesNotExist**

Get the list of all monitors.

```
String[][] getMonitors(
    String[] names
)
```

**getMonitorsByLocation( location, names ) throws ObjectDoesNotExist**

Get the list of all monitors. This is a location specific function, any action will operate on the specified location.

```
String[][] getMonitorsByLocation(
    String location
    String[] names
)
```

**getNodeCloseWithRst( names ) throws ObjectDoesNotExist**

Get whether connections to the back-end nodes should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state.

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

**getNodeCloseWithRstByLocation( location, names ) throws ObjectDoesNotExist**

Get whether connections to the back-end nodes should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getNodeCloseWithRstByLocation(
    String location
    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
)

getNodeConnCloseByLocation( location, names ) throws ObjectDoesNotExist

Get whether all connections that have been sent to a node are closed when that node is marked as dead. This is a location specific function, any action will operate on the specified location.

Boolean[] getNodeConnCloseByLocation(
    String location
    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 when passive monitoring is enabled.

Unsigned Integer[] getNodeConnectionAttempts(
    String[] names
)

getNodeConnectionAttemptsByLocation( location, names ) throws ObjectDoesNotExist

Get the number of times your traffic manager should try and connect to a node before registering it as failed when passive monitoring is enabled. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getNodeConnectionAttemptsByLocation(
    String location
    String[] names
)

getNodeDeleteBehavior( names ) throws ObjectDoesNotExist

Get the deletion behavior for nodes in a pool.

Pool.NodeDeleteBehavior[] getNodeDeleteBehavior(
    String[] names
)

getNodeDeleteBehaviorByLocation( location, names ) throws ObjectDoesNotExist

Get the deletion behavior for nodes in a pool. This is a location specific function, any action will operate on the specified location.

Pool.NodeDeleteBehavior[] getNodeDeleteBehaviorByLocation(
    String location
    String[] names
)
**getNodeDrainToDeleteTimeout( names ) throws ObjectDoesNotExist**

Get the maximum time that a node will be allowed to remain in a draining state after it has been deleted. A value of 0 means no maximum time.

```java
Unsigned Integer[] getNodeDrainToDeleteTimeout(
    String[] names
)
```

**getNodeDrainToDeleteTimeoutByLocation( location, names ) throws ObjectDoesNotExist**

Get the maximum time that a node will be allowed to remain in a draining state after it has been deleted. A value of 0 means no maximum time. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getNodeDrainToDeleteTimeoutByLocation(
    String location,
    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

```java
Unsigned Integer[] getNodeFailTime(
    String[] names
)
```

**getNodeFailTimeByLocation( location, names ) throws ObjectDoesNotExist**

Get the length of time a failed node should be isolated for before testing it with new traffic, in seconds. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getNodeFailTimeByLocation(
    String location,
    String[] names
)
```

**getNodeUseNagle( names ) throws ObjectDoesNotExist**

Get whether Nagle’s algorithm should be used for TCP connections to the back-end nodes.

```java
Boolean[] getNodeUseNagle(
    String[] names
)
```

**getNodeUseNagleByLocation( location, names ) throws ObjectDoesNotExist**

Get whether Nagle’s algorithm should be used for TCP connections to the back-end nodes. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getNodeUseNagleByLocation(
    String location,
    String[] names
)
```

**getNodes( names ) throws ObjectDoesNotExist**

Get the lists of nodes for each of the named pools.
getNodesByLocation( location, names ) throws ObjectDoesNotExist

Get the lists of nodes for each of the named pools. This is a location specific function, any action will operate on the specified location.

getNodesConnectionCounts( nodes )

Get the number of active connections to each of the specified nodes.

getNodesLastUsed( nodes )

Get the number of seconds since each of the specified nodes was last used.

getNodesPriorityValue( names, nodes ) throws ObjectDoesNotExist

For each of the named pools, get the priority values for the named nodes in each pool.

getNodesPriorityValueByLocation( location, names, nodes ) throws ObjectDoesNotExist

For each of the named pools, get the priority values for the named nodes in each pool. This is a location specific function, any action will operate on the specified location.

getNodesWeightings( names, nodes ) throws ObjectDoesNotExist

For each of the named pools, get the weighting values for the specified nodes in this pool.
Function Reference

**getNodesWeightingsByLocation**

getNodesWeightingsByLocation( location, names, nodes ) throws ObjectDoesNotExist

For each of the named pools, get the weighting values for the specified nodes in this pool. This is a location specific function, any action will operate on the specified location.

```
Pool.WeightingsDefinition[][] getNodesWeightingsByLocation(
    String location
    String[] names
    String[][] nodes
)
```

**getNote**

getNote( names ) throws ObjectDoesNotExist

Get the note for each of the named pools.

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

**getPassiveMonitoring**

getPassiveMonitoring( names ) throws ObjectDoesNotExist

Get whether this pool uses passive monitoring.

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

**getPassiveMonitoringByLocation**

getPassiveMonitoringByLocation( location, names ) throws ObjectDoesNotExist

Get whether this pool uses passive monitoring. This is a location specific function, any action will operate on the specified location.

```
Boolean[] getPassiveMonitoringByLocation(
    String location
    String[] names
)
```

**getPersistence**

getPersistence( names ) throws ObjectDoesNotExist

Get the default Session Persistence classes that each of the named pools uses.

```
String[] getPersistence(
    String[] names
)
```

**getPersistenceByLocation**

getPersistenceByLocation( location, names ) throws ObjectDoesNotExist

Get the default Session Persistence classes that each of the named pools uses. This is a location specific function, any action will operate on the specified location.

```
String[] getPersistenceByLocation(
    String location
    String[] names
)
```

**getPoolNames**

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
)

getPriorityEnabledByLocation(location, names) throws ObjectDoesNotExist

Get whether each of the named pools uses priority lists. This is a location specific function, any action will operate on the specified location.

Boolean[] getPriorityEnabledByLocation(
    String location
    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
)

getPriorityNodesByLocation(location, names) throws ObjectDoesNotExist

Get the minimum number of highest-priority active nodes, for each of the named pools. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getPriorityNodesByLocation(
    String location
    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
)

getPriorityValuesByLocation(location, names) throws ObjectDoesNotExist

For each of the named pools, get the priority values for each of the nodes in each pool. This is a location specific function, any action will operate on the specified location.

Pool.PriorityValueDefinition[][] getPriorityValuesByLocation(
    String location
    String[] names
)

gQueueTimeout(names) throws ObjectDoesNotExist

Get is the maximum time to keep a connections queued in seconds. A value of 0 will not timeout queued connections.

Unsigned Integer[] getQueueTimeout(
    String[] names
)
getQueueTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get is the maximum time to keep a connections queued in seconds. A value of 0 will not timeout queued connections. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getQueueTimeoutByLocation(
    String location
    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
)

getSMTPSendStartTLSByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

Boolean[] getSMTPSendStartTLSByLocation(
    String location
    String[] names
)

getSSLCiphers( names ) throws ObjectDoesNotExist

Get the ciphers allowed for connection to a back-end node

String[] getSSLCiphers(
    String[] names
)

getSSLCiphersByLocation( location, names ) throws ObjectDoesNotExist

Get the ciphers allowed for connection to a back-end node This is a location specific function, any action will operate on the specified location.

String[] getSSLCiphersByLocation(
    String location
    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
)
getSSLClientAuthByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Boolean[] getSSLClientAuthByLocation(
    String location
    String[] names
)

getSSLCommonNameMatch( names ) throws ObjectDoesNotExist

Get the list of names against which the 'common name' of the certificate is matched.

String[][] getSSLCommonNameMatch(
    String[] names
)

getSSLCommonNameMatchByLocation( location, names ) throws ObjectDoesNotExist

Get the list of names against which the 'common name' of the certificate is matched. This is a location specific function, any action will operate on the specified location.

String[][] getSSLCommonNameMatchByLocation(
    String location
    String[] names
)

getSSLEllipticCurves( names ) throws ObjectDoesNotExist

Get the elliptic curve preference list for SSL connections from this pool

String[] getSSLEllipticCurves(
    String[] names
)

getSSLEllipticCurvesByLocation( location, names ) throws ObjectDoesNotExist

Get the elliptic curve preference list for SSL connections from this pool. This is a location specific function, any action will operate on the specified location.

String[] getSSLEllipticCurvesByLocation(
    String location
    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
)
**getSSLEncryptByLocation( location, names ) throws ObjectDoesNotExist**

Get whether each of the named pools should encrypt data to the back-end nodes using SSL. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLEncryptByLocation(
    String location,
    String[] names
)
```

**getSSLEnhance( names ) throws ObjectDoesNotExist**

Get whether each of the named pools should use SSL protocol enhancements. These enhancements allow Brocade vTM virtual servers to discover the original client’s IP address. Only use enable this if, for this pool, you are using Brocade Virtual Traffic Managers whose virtual servers have the ‘ssl_trust_magic’ setting enabled.

```java
Boolean[] getSSLEnhance(
    String[] names
)
```

**getSSLEnhanceByLocation( location, names ) throws ObjectDoesNotExist**

Get whether each of the named pools should use SSL protocol enhancements. These enhancements allow Brocade vTM virtual servers to discover the original client’s IP address. Only use enable this if, for this pool, you are using Brocade Virtual Traffic Managers whose virtual servers have the ‘ssl_trust_magic’ setting enabled. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLEnhanceByLocation(
    String location,
    String[] names
)
```

**getSSLSendCloseAlerts( names ) throws ObjectDoesNotExist**

Get whether each of the named pools should send a close alert when they initiate socket disconnections.

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

**getSSLSendCloseAlertsByLocation( location, names ) throws ObjectDoesNotExist**

Get whether each of the named pools should send a close alert when they initiate socket disconnections. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLSendCloseAlertsByLocation(
    String location,
    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.

```java
Boolean[] getSSLServerNameExtension(~
    String[] names
)
```
getSSLServerNameExtensionByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLServerNameExtensionByLocation(
    String location
    String[] names
)
```

getSSLSignatureAlgorithms( names ) throws ObjectDoesNotExist

Get the SSL signature algorithms preference list for SSL connections from this pool

```java
String[] getSSLSignatureAlgorithms(
    String[] names
)
```

getSSLSignatureAlgorithmsByLocation( location, names ) throws ObjectDoesNotExist

Get the SSL signature algorithms preference list for SSL connections from this pool. This is a location specific function, any action will operate on the specified location.

```java
String[] getSSLSignatureAlgorithmsByLocation(
    String location
    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.

```java
Boolean[] getSSLStrictVerify(
    String[] names
)
```

getSSLStrictVerifyByLocation( location, names ) throws ObjectDoesNotExist

Get whether each of the named pools should perform strict certificate validation on SSL certificates from the back-end nodes. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getSSLStrictVerifyByLocation(
    String location
    String[] names
)
```

getSSLSupportSSL2( names ) throws ObjectDoesNotExist

This method is now deprecated.

```java
Pool.SSLSupportSSL2[] getSSLSupportSSL2(
    String[] names
)
```
getSSLSupportSSL2ByLocation( location, names ) throws ObjectDoesNotExist
This method is now deprecated. This is a location specific function, any action will operate on the specified location.

```
Pool.SSLSupportSSL2[] getSSLSupportSSL2ByLocation(
    String location
    String[] names
)
```

getSSLSupportSSL3( names ) throws ObjectDoesNotExist
Get whether SSLv3 is enabled for this pool

```
Pool.SSLSupportSSL3[] getSSLSupportSSL3(
    String[] names
)
```

getSSLSupportSSL3ByLocation( location, names ) throws ObjectDoesNotExist
Get whether SSLv3 is enabled for this pool This is a location specific function, any action will operate on the specified location.

```
Pool.SSLSupportSSL3[] getSSLSupportSSL3ByLocation(
    String location
    String[] names
)
```

getSSLSupportTLS1( names ) throws ObjectDoesNotExist
Get whether TLSv1.0 is enabled for this pool

```
Pool.SSLSupportTLS1[] getSSLSupportTLS1(
    String[] names
)
```

getSSLSupportTLS11( names ) throws ObjectDoesNotExist
Get whether TLSv1.1 is enabled for this pool

```
Pool.SSLSupportTLS11[] getSSLSupportTLS11(
    String[] names
)
```

getSSLSupportTLS11ByLocation( location, names ) throws ObjectDoesNotExist
Get whether TLSv1.1 is enabled for this pool This is a location specific function, any action will operate on the specified location.

```
Pool.SSLSupportTLS11[] getSSLSupportTLS11ByLocation(
    String location
    String[] names
)
```

getSSLSupportTLS12( names ) throws ObjectDoesNotExist
Get whether TLSv1.2 is enabled for this pool

```
Pool.SSLSupportTLS12[] getSSLSupportTLS12(
    String[] names
)
```
getSSLSupportTLS12ByLocation( location, names ) throws ObjectDoesNotExist

Get whether TLSv1.2 is enabled for this pool. This is a location specific function, any action will operate on the specified location.

```java
Pool.SSLSupportTLS12[] getSSLSupportTLS12ByLocation(
    String location
    String[] names
)
```

getSSLSupportTLS1ByLocation( location, names ) throws ObjectDoesNotExist

Get whether TLSv1.0 is enabled for this pool. This is a location specific function, any action will operate on the specified location.

```java
Pool.SSLSupportTLS1[] getSSLSupportTLS1ByLocation(
    String location
    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.

```java
Boolean[] getTransparent(
    String[] names
)
```

getTransparentByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getTransparentByLocation(
    String location
    String[] names
)
```

getUDPAcceptFrom( names ) throws ObjectDoesNotExist

Get what sets of IP addresses and ports from which we should accept UDP responses.

```java
Pool.UDPAcceptFrom[] getUDPAcceptFrom(
    String[] names
)
```

getUDPAcceptFromByLocation( location, names ) throws ObjectDoesNotExist

Get what sets of IP addresses and ports from which we should accept UDP responses. This is a location specific function, any action will operate on the specified location.

```java
Pool.UDPAcceptFrom[] getUDPAcceptFromByLocation(
    String location
    String[] names
)
```

getUDPAcceptFromIPMask( names ) throws ObjectDoesNotExist

Get the mask to validate the IP of UDP responses with. Only used if UDPAcceptFromIP is set to 'ip_mask'.
Function Reference

String[] getUDPAcceptFromIPMask{
    String[] names
}

getUDPAcceptFromIPMaskByLocation( location, names ) throws ObjectDoesNotExist

Get the mask to validate the IP of UDP responses with. Only used if UDPAcceptFromIP is set to 'ip_mask'. This is a location specific function, any action will operate on the specified location.

String[] getUDPAcceptFromIPMaskByLocation{
    String location
    String[] names
}

getUDPResponseTimeout( names ) throws ObjectDoesNotExist

Get the maximum delay in replying before a node receiving UDP packets is assumed to have failed.

Unsigned Integer[] getUDPResponseTimeout{
    String[] names
}

getUDPResponseTimeoutByLocation( location, names ) throws ObjectDoesNotExist

Get the maximum delay in replying before a node receiving UDP packets is assumed to have failed. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getUDPResponseTimeoutByLocation{
    String location
    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
}

getWeightingsByLocation( location, names ) throws ObjectDoesNotExist

For each of the named pools, get the weightings for each of the nodes in each pool. This is a location specific function, any action will operate on the specified location.

Pool.WeightingsDefinition[][] getWeightingsByLocation{
    String location
    String[] names
}

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

Remove the security group IDs to associate to the new EC2 instances.

void removeAutoscaleSecuritygroupids{
    String[] names
    String[][] values
}
removeAutoscaleSecuritygroupidsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove the security group IDs to associate to the new EC2 instances. This is a location specific function, any action will operate on the specified location.

```java
void removeAutoscaleSecuritygroupidsByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Remove the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic.

```java
void removeAutoscaleSubnetids(
    String[] names
    String[][] values
)
```

removeAutoscaleSubnetidsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic. This is a location specific function, any action will operate on the specified location.

```java
void removeAutoscaleSubnetidsByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Remove the hostnames to be used for DNS-derived autoscaling

```java
void removeDNSAutoscaleHostnames(
    String[] names
    String[][] values
)
```

removeDNSAutoscaleHostnamesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove the hostnames to be used for DNS-derived autoscaling This is a location specific function, any action will operate on the specified location.

```java
void removeDNSAutoscaleHostnamesByLocation(
    String location
    String[] names
    String[][] values
)
```
Function Reference

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

Remove nodes from the lists of draining nodes, for each of the named pools.

```java
void removeDrainingNodes{
    String[] names
    String[][] values
}
```

removeDrainingNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove nodes from the lists of draining nodes, for each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void removeDrainingNodesByLocation{
    String location
    String[] names
    String[][] values
}
```

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

Remove monitors from each of the named pools.

```java
void removeMonitors{
    String[] names
    String[][] values
}
```

removeMonitorsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove monitors from each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void removeMonitorsByLocation{
    String location
    String[] names
    String[][] values
}
```

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

Remove nodes from each of the named pools.

```java
void removeNodes{
    String[] names
    String[][] values
}
```

removeNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove nodes from each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void removeNodesByLocation{
    String location
    String[] names
    String[][] values
}
Pool

void removeNodesByLocation(
    String location
    String[] names
    String[][] values
)

removeSSLCommonNameMatch( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Remove the list of names against which the 'common name' of the certificate is matched.

void removeSSLCommonNameMatch(
    String[] names
    String[][] values
)

removeSSLCommonNameMatchByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Remove the list of names against which the 'common name' of the certificate is matched. This is a location specific function, any action will operate on the specified location.

void removeSSLCommonNameMatchByLocation(
    String location
    String[] names
    String[][] values
)

crenamePool( names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation
Rename each of the named pools.

void renamePool(
    String[] names
    String[] new_names
)

setAutoscaleAddnodeDelaytime( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set Delay in seconds before the node should be added to the autoscaled pool

void setAutoscaleAddnodeDelaytime(
    String[] names
    Unsigned Integer[] values
)

setAutoscaleAddnodeDelaytimeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set Delay in seconds before the node should be added to the autoscaled pool This is a location specific function, any action will operate on the specified location.

void setAutoscaleAddnodeDelaytimeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
setAutoscaleCloudcredentials( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the cloud credentials for this autoscaled pool.

```java
void setAutoscaleCloudcredentials(
    String[] names
    String[] values
)
```

setAutoscaleCloudcredentialsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the cloud credentials for this autoscaled pool. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleCloudcredentialsByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set the ESX host or ESX cluster name to put the new virtual machine instances on.

```java
void setAutoscaleCluster(
    String[] names
    String[] values
)
```

setAutoscaleClusterByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the ESX host or ESX cluster name to put the new virtual machine instances on. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleClusterByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set the name of the logical datacenter on the vCenter server.

```java
void setAutoscaleDatacenter(
    String[] names
    String[] values
)
```

setAutoscaleDatacenterByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the name of the logical datacenter on the vCenter server. This is a location specific function, any action will operate on the specified location.
void setAutoscaleDatacenterByLocation{
    String location
    String[] names
    String[] values
}

setAutoscaleDatastore( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set The name of the datastore to be used by the newly created virtual machine.

void setAutoscaleDatastore{
    String[] names
    String[] values
}

setAutoscaleDatastoreByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set The name of the datastore to be used by the newly created virtual machine. This is a location specific function, any action will operate on the specified location.

void setAutoscaleDatastoreByLocation{
    String location
    String[] names
    String[] values
}

setAutoscaleEnabled( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set whether this pool uses autoscaling.

void setAutoscaleEnabled{
    String[] names
    Boolean[] values
}

setAutoscaleEnabledByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set whether this pool uses autoscaling. This is a location specific function, any action will operate on the specified location.

void setAutoscaleEnabledByLocation{
    String location
    String[] names
    Boolean[] values
}

setAutoscaleExternal( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set whether autoscaling is handled externally or internally

void setAutoscaleExternal{
    String[] names
    Boolean[] values
}
setAutoscaleExternalByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether autoscaling is handled externally or internally. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleExternalByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set any extra arguments to the autoscaling API. Each argument can be separated by comma. E.g. in case of EC2, it can take extra parameters to the Amazon's RunInstance API say DisableApiTermination=false,Placement.Tenancy=default.

```java
void setAutoscaleExtraargs(
    String[] names
    String[] values
)
```

setAutoscaleExtraargsByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set any extra arguments to the autoscaling API. Each argument can be separated by comma. E.g. in case of EC2, it can take extra parameters to the Amazon's RunInstance API say DisableApiTermination=false,Placement.Tenancy=default. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleExtraargsByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set the hysteresis period for an autoscaled pool.

```java
void setAutoscaleHysteresis(
    String[] names
    Unsigned Integer[] values
)
```

setAutoscaleHysteresisByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the hysteresis period for an autoscaled pool. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleHysteresisByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```
setAutoscaleImageid( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the image identifier

```java
void setAutoscaleImageid(
    String[] names
    String[] values
)
```

setAutoscaleImageidByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the image identifier This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleImageidByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set whether to use the public or private IPs

```java
void setAutoscaleIpstouse(
    String[] names
    Pool.AutoscaleIpstouse[] values
)
```

setAutoscaleIpstouseByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether to use the public or private IPs This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleIpstouseByLocation(
    String location
    String[] names
    Pool.AutoscaleIpstouse[] values
)
```

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

Set the idle time of the last node in an autoscaled pool before it can be destroyed

```java
void setAutoscaleLastnodeIdletime(
    String[] names
    Unsigned Integer[] values
)
```

setAutoscaleLastnodeIdletimeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the idle time of the last node in an autoscaled pool before it can be destroyed This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleLastnodeIdletimeByLocation(
```

```java
```
Function Reference

Pool

String location
String[] names
Unsigned Integer[] values

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

Set the maximum number of nodes in an autoscaled pool

void setAutoscaleMaxNodes(
    String[] names
    Unsigned Integer[] values
)

setAutoscaleMaxNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum number of nodes in an autoscaled pool This is a location specific function, any action will operate on the specified location.

void setAutoscaleMaxNodesByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

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

Set the minimum number of nodes in an autoscaled pool

void setAutoscaleMinNodes(
    String[] names
    Unsigned Integer[] values
)

setAutoscaleMinNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the minimum number of nodes in an autoscaled pool This is a location specific function, any action will operate on the specified location.

void setAutoscaleMinNodesByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

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

Set the node name prefix for this autoscaled pool

void setAutoscaleName(
    String[] names
    String[] values
)
**setAutoscaleNameByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the node name prefix for this autoscaled pool. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleNameByLocation(
    String location
    String[] names
    String[] values
)
```

**setAutoscalePort( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the port number for this autoscaled pool.

```java
void setAutoscalePort(
    String[] names
    Unsigned Integer[] values
)
```

**setAutoscalePortByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the port number for this autoscaled pool. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscalePortByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setAutoscaleRefractory( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the refractory period for an autoscaled pool.

```java
void setAutoscaleRefractory(
    String[] names
    Unsigned Integer[] values
)
```

**setAutoscaleRefractoryByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the refractory period for an autoscaled pool. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleRefractoryByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```
Function Reference

### setAutoscaleResponseTime( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the expected node response time in milliseconds

```java
void setAutoscaleResponseTime(
    String[] names
    Unsigned Integer[] values
)
```

### setAutoscaleResponseTimeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the expected node response time in milliseconds This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleResponseTimeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

### setAutoscaleScaledownLevel( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the threshold of conforming requests for scaling down

```java
void setAutoscaleScaledownLevel(
    String[] names
    Unsigned Integer[] values
)
```

### setAutoscaleScaledownLevelByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the threshold of conforming requests for scaling down This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleScaledownLevelByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

### setAutoscaleScaleupLevel( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the acceptable lower percentage of conforming requests

```java
void setAutoscaleScaleupLevel(
    String[] names
    Unsigned Integer[] values
)
```

### setAutoscaleScaleupLevelByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the acceptable lower percentage of conforming requests This is a location specific function, any action will operate on the specified location.

```java
```
void setAutoscaleScaleupLevelByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)

setAutoscaleSecuritygroupids( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the security group IDs to associate to the new EC2 instances.

void setAutoscaleSecuritygroupids( 
    String[] names
    String[][] values
)

setAutoscaleSecuritygroupidsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the security group IDs to associate to the new EC2 instances. This is a location specific function, any action will operate on the specified location.

void setAutoscaleSecuritygroupidsByLocation(
    String location
    String[] names
    String[][] values
)

setAutoscaleSizeid( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the size identifier

void setAutoscaleSizeid( 
    String[] names
    String[] values
)

setAutoscaleSizeidByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the size identifier This is a location specific function, any action will operate on the specified location.

void setAutoscaleSizeidByLocation( 
    String location
    String[] names
    String[] values
)

setAutoscaleSubnetids( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic.

void setAutoscaleSubnetids( 
    String[] names
    String[][] values
)
setAutoscaleSubnetidsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the list of subnet IDs where the new EC2-VPC instances will be launched. Instances will be evenly distributed among the subnets. If the list is empty, instances will be launched inside EC2-Classic. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscaleSubnetidsByLocation(
    String location,
    String[] names,
    String[][] values
)
```

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

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

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

setBandwidthClassByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the Bandwidth Classes that each of the named pools uses. This is a location specific function, any action will operate on the specified location.

```java
void setBandwidthClassByLocation(
    String location,
    String[] names,
    String[] values
)
```

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

Set whether this pool uses DNS-derived autoscaling

```java
void setDNSAutoscaleEnabled(
    String[] names,
    Boolean[] values
)
```

setDNSAutoscaleEnabledByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether this pool uses DNS-derived autoscaling. This is a location specific function, any action will operate on the specified location.

```java
void setDNSAutoscaleEnabledByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```
setDNSAutoscaleHostnames( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the hostnames to be used for DNS-derived autoscaling

```java
void setDNSAutoscaleHostnames(
    String[] names
    String[][] values
)
```

setDNSAutoscaleHostnamesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the hostnames to be used for DNS-derived autoscaling This is a location specific function, any action will operate on the specified location.

```java
void setDNSAutoscaleHostnamesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Set the port number for DNS-derived autoscaling in this pool

```java
void setDNSAutoscalePort(
    String[] names
    Unsigned Integer[] values
)
```

setDNSAutoscalePortByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the port number for DNS-derived autoscaling in this pool This is a location specific function, any action will operate on the specified location.

```java
void setDNSAutoscalePortByLocation(
    String location
    String[] names
    Unsigned Integer[] 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 remain in their existing state).

```java
void setDisabledNodes(
    String[] names
    String[][] nodes
)
```
**setDisabledNodesByLocation( location, 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 remain in their existing state). This is a location specific function, any action will operate on the specified location.

```java
void setDisabledNodesByLocation(
    String location
    String[] names
    String[][] nodes
)
```

**setDrainingNodes( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the lists of draining nodes for each of the named pools.

```java
void setDrainingNodes(
    String[] names
    String[][] values
)
```

**setDrainingNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the lists of draining nodes for each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void setDrainingNodesByLocation(
    String location
    String[] names
    String[][] values
)
```

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

This method is now obsolete and is replaced by VirtualServer.setErrorFile.

```java
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.

```java
void setFTPSupportRfc2428(
    String[] names
    Boolean[] values
)
```

**setFTPSupportRfc2428ByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set whether backend IPv4 nodes understand the FTP EPRT and EPSV commands. This is a location specific function, any action will operate on the specified location.

```java
void setFTPSupportRfc2428ByLocation(
    String location
    String[] names
    Boolean[] values
)
void setFTPSupportRfc2428ByLocation(
    String location
    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
)

setFailpoolByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void setFailpoolByLocation(
    String location
    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
)

setKeepaliveByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should maintain HTTP keepalive connections to the nodes. This is a location specific function, any action will operate on the specified location.

void setKeepaliveByLocation(
    String location
    String[] names
    Boolean[] values
)

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

Set whether each of the named pools should maintain HTTP keepalive connections to the nodes for non-idempotent requests.

void setKeepaliveNonIdempotent(
    String[] names
    Boolean[] values
)
setKeepaliveNonIdempotentByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should maintain HTTP keepalive connections to the nodes for non-idempotent requests. This is a location specific function, any action will operate on the specified location.

```java
void setKeepaliveNonIdempotentByLocation(
    String location,
    String[] names,
    Boolean[] values
)
```

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

Set the Kerberos principal that each of the named pools uses to perform Kerberos Protocol Transition

```java
void setKerberosProtocolTransitionPrincipal(
    String[] names,
    String[] values
)
```

setKerberosProtocolTransitionPrincipalByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the Kerberos principal that each of the named pools uses to perform Kerberos Protocol Transition This is a location specific function, any action will operate on the specified location.

```java
void setKerberosProtocolTransitionPrincipalByLocation(
    String location,
    String[] names,
    String[] values
)
```

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

Set the Kerberos principal name of the service that each of the named pools target

```java
void setKerberosProtocolTransitionTarget(
    String[] names,
    String[] values
)
```

setKerberosProtocolTransitionTargetByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the Kerberos principal name of the service that each of the named pools target This is a location specific function, any action will operate on the specified location.

```java
void setKerberosProtocolTransitionTargetByLocation(
    String location,
    String[] names,
    String[] values
)
```
**setL4AccelSNAT( names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether connections to the back-end nodes should appear to originate from an IP address raised on the traffic manager, rather than the IP address from which they were received by the traffic manager.

```java
void setL4AccelSNAT(
    String[] names
    Boolean[] values
)
```

**setL4AccelSNATByLocation( location, names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether connections to the back-end nodes should appear to originate from an IP address raised on the traffic manager, rather than the IP address from which they were received by the traffic manager. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelSNATByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setLoadBalancingAlgorithm( names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

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

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

**setLoadBalancingAlgorithmByLocation( location, names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the load balancing algorithms that each of the named pools uses. This is a location specific function, any action will operate on the specified location.

```java
void setLoadBalancingAlgorithmByLocation(
    String location
    String[] names
    Pool.LoadBalancingAlgorithm[] values
)
```

**setMaxConnectTime( names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the time that each of the named pools should wait for a connection to establish to a node before trying another node, in seconds.

```java
void setMaxConnectTime(
    String[] names
    Unsigned Integer[] values
)
```
setMaxConnectTimeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the time that each of the named pools should wait for a connection to establish to a node before trying another node, in seconds. This is a location specific function, any action will operate on the specified location.

```
void setMaxConnectTimeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set the number of times that each of the named pools can try to connect to any of its nodes before sending an error response.

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

setMaxConnectionAttemptsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the number of times that each of the named pools can try to connect to any of its nodes before sending an error response. This is a location specific function, any action will operate on the specified location.

```
void setMaxConnectionAttemptsByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set is the maximum number of concurrent connections allowed to each node in the pool per machine.

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

setMaxConnectionsPernodeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set is the maximum number of concurrent connections allowed to each node in the pool per machine. This is a location specific function, any action will operate on the specified location.

```
void setMaxConnectionsPernodeByLocation(
    String location
    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.

```java
void setMaxIdleConnectionsPerNode(
    String[] names
    Unsigned Integer[] values
)
```

setMaxIdleConnectionsPerNodeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setMaxIdleConnectionsPerNodeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

setMaxKeepalivesPerNode is deprecated, please use setMaxIdleConnectionsPerNode instead.

```java
void setMaxKeepalivesPerNode(
    String[] names
    Unsigned Integer[] values
)
```

setMaxKeepalivesPerNodeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

setMaxKeepalivesPerNode is deprecated, please use setMaxIdleConnectionsPerNode instead. This is a location specific function, any action will operate on the specified location.

```java
void setMaxKeepalivesPerNodeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set is the maximum number of connections that can be queued due to connection limits.

```java
void setMaxQueueSize(
    String[] names
    Unsigned Integer[] values
)
```
setMaxQueueSizeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set is the maximum number of connections that can be queued due to connection limits. This is a location specific function, any action will operate on the specified location.

```
void setMaxQueueSizeByLocation(
    String location
    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
)
```

setMaxReplyTimeByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

```
void setMaxReplyTimeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set the number of times that each of the named pools can try to connect and time out waiting for a response, by exceeding max_reply_time, to any of its nodes before sending an error response.

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

setMaxTimedOutConnectionAttemptsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the number of times that each of the named pools can try to connect and time out waiting for a response, by exceeding max_reply_time, to any of its nodes before sending an error response. This is a location specific function, any action will operate on the specified location.

```
void setMaxTimedOutConnectionAttemptsByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```
setMonitors( names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

Set the list of all monitors.

```java
void setMonitors(
    String[] names
    String[][] values
)
```

setMonitorsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError

Set the list of all monitors. This is a location specific function, any action will operate on the specified location.

```java
void setMonitorsByLocation(
    String location
    String[] names
    String[][] values
)
```

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

Set whether connections to the back-end nodes should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state.

```java
void setNodeCloseWithRst(
    String[] names
    Boolean[] values
)
```

setNodeCloseWithRstByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether connections to the back-end nodes should be closed with a RST packet, rather than a FIN packet, avoiding the TIME_WAIT state. This is a location specific function, any action will operate on the specified location.

```java
void setNodeCloseWithRstByLocation(
    String location
    String[] names
    Boolean[] 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.

```java
void setNodeConnClose(
    String[] names
    Boolean[] values
)
**setNodeConnCloseByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setNodeConnCloseByLocation(
    String location
    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 when passive monitoring is enabled.

```java
void setNodeConnectionAttempts(
    String[] names
    Unsigned Integer[] values
)
```

**setNodeConnectionAttemptsByLocation( location, 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 when passive monitoring is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setNodeConnectionAttemptsByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setNodeDeleteBehavior( names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the deletion behavior for nodes in a pool.

```java
void setNodeDeleteBehavior(
    String[] names
    Pool.NodeDeleteBehavior[] values
)
```

**setNodeDeleteBehaviorByLocation( location, names, values )** throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the deletion behavior for nodes in a pool. This is a location specific function, any action will operate on the specified location.

```java
void setNodeDeleteBehaviorByLocation(
    String location
    String[] names
    Pool.NodeDeleteBehavior[] values
)
```
setNodeDrainToDeleteTimeout( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum time that a node will be allowed to remain in a draining state after it has been deleted. A value of 0 means no maximum time.

```java
void setNodeDrainToDeleteTimeout(
    String[] names
    Unsigned Integer[] values
)
```

setNodeDrainToDeleteTimeoutByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum time that a node will be allowed to remain in a draining state after it has been deleted. A value of 0 means no maximum time. This is a location specific function, any action will operate on the specified location.

```java
void setNodeDrainToDeleteTimeoutByLocation(
    String location
    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

```java
void setNodeFailTime(
    String[] names
    Unsigned Integer[] values
)
```

setNodeFailTimeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setNodeFailTimeByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

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

Set whether Nagle's algorithm should be used for TCP connections to the back-end nodes.

```java
void setNodeUseNagle(
    String[] names
    Boolean[] values
)
```
setNodeUseNagleByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether Nagle's algorithm should be used for TCP connections to the back-end nodes. This is a location specific function, any action will operate on the specified location.

```java
void setNodeUseNagleByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set the lists of nodes for each of the named pools.

```java
void setNodes{
    String[] names
    String[][] values
}
```

setNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the lists of nodes for each of the named pools. This is a location specific function, any action will operate on the specified location.

```java
void setNodesByLocation{
    String location
    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.

```java
void setNodesPriorityValue{
    String[] names
    Pool.PriorityValueDefinition[][] node_values
}
```

setNodesPriorityValueByLocation( location, names, node_values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

For each of the named pools, set the priority values for the named nodes in each pool. This is a location specific function, any action will operate on the specified location.

```java
void setNodesPriorityValueByLocation{
    String location
    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
)

setNodesWeightingsByLocation(location, 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. This is a location specific function, any action will operate on the specified location.

void setNodesWeightingsByLocation(
    String location
    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
)

setPassiveMonitoringByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether this pool uses passive monitoring. This is a location specific function, any action will operate on the specified location.

void setPassiveMonitoringByLocation(
    String location
    String[] names
    Boolean[] values
)

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

Set the default Session Persistence classes that each of the named pools uses.
Function Reference

void setPersistence{
    String[] names
    String[] values
}

setPersistenceByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the default Session Persistence classes that each of the named pools uses. This is a location specific function, any action will operate on the specified location.

void setPersistenceByLocation{
    String location
    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
}

setPriorityEnabledByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools uses priority lists. This is a location specific function, any action will operate on the specified location.

void setPriorityEnabledByLocation{
    String location
    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
}

setPriorityNodesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the minimum number of highest-priority active nodes, for each of the named pools. This is a location specific function, any action will operate on the specified location.

void setPriorityNodesByLocation{
    String location
    String[] names
    Unsigned Integer[] values
}
setQueueTimeout( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum time to keep connections queued in seconds. A value of 0 will not timeout queued connections.

```java
void setQueueTimeout(
    String[] names
    Unsigned Integer[] values
)
```

setQueueTimeoutByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum time to keep connections queued in seconds. A value of 0 will not timeout queued connections. This is a location specific function, any action will operate on the specified location.

```java
void setQueueTimeoutByLocation(
    String location
    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).

```java
void setSMTPSendStartTLS(
    String[] names
    Boolean[] values
)
```

setSMTPSendStartTLSByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

```java
void setSMTPSendStartTLSByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set the ciphers allowed for connection to a back-end node.

```java
void setSSLCiphers(
    String[] names
    String[] values
)
```
setSSLciphersByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the ciphers allowed for connection to a back-end node. This is a location specific function, any action will operate on the specified location.

```java
void setSSLciphersByLocation(
    String location
    String[] names
    String[] 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.

```java
void setSSLClientAuth(
    String[] names
    Boolean[] values
)
```

setSSLClientAuthByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSSLClientAuthByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set the list of names against which the 'common name' of the certificate is matched.

```java
void setSSLCommonNameMatch(
    String[] names
    String[][] values
)
```

setSSLCommonNameMatchByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the list of names against which the 'common name' of the certificate is matched. This is a location specific function, any action will operate on the specified location.

```java
void setSSLCommonNameMatchByLocation(
    String location
    String[] names
    String[][] values
)
```
setSSLEllipticCurves( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the elliptic curve preference list for SSL connections from this pool

```java
void setSSLEllipticCurves(
    String[] names
    String[] values
)
```

setSSLEllipticCurvesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the elliptic curve preference list for SSL connections from this pool This is a location specific function, any action will operate on the specified location.

```java
void setSSLEllipticCurvesByLocation(
    String location
    String[] names
    String[] 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.

```java
void setSSLEncrypt(
    String[] names
    Boolean[] values
)
```

setSSLEncryptByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should encrypt data to the back-end nodes using SSL. This is a location specific function, any action will operate on the specified location.

```java
void setSSLEncryptByLocation(
    String location
    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 Brocade vTM virtual servers to discover the original client's IP address. Only use enable this if, for this pool, you are using Brocade Virtual Traffic Managers whose virtual servers have the 'ssl_trust_magic' setting enabled.

```java
void setSSLEnhance(
    String[] names
    Boolean[] values
)
setSSLEnhanceByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should use SSL protocol enhancements. These enhancements allow Brocade vTM virtual servers to discover the original client's IP address. Only use enable this if, for this pool, you are using Brocade Virtual Traffic Managers whose virtual servers have the 'ssl_trust_magic' setting enabled. This is a location specific function, any action will operate on the specified location.

```java
void setSSLEnhanceByLocation(
    String location
    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.

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

setSSLSendCloseAlertsByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether each of the named pools should send a close alert when they initiate socket disconnections. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSendCloseAlertsByLocation(
    String location
    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.

```java
void setSSLServerNameExtension(
    String[] names
    Boolean[] values
)
```

setSSLServerNameExtensionByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSSLServerNameExtensionByLocation(
    String location
    String[] names
    Boolean[] values
)
```
setSSLSignatureAlgorithms(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SSL signature algorithms preference list for SSL connections from this pool

```java
void setSSLSignatureAlgorithms(
    String[] names
    String[] values
)
```

setSSLSignatureAlgorithmsByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SSL signature algorithms preference list for SSL connections from this pool. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSignatureAlgorithmsByLocation(
    String location
    String[] names
    String[] 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.

```java
void setSSLStrictVerify(
    String[] names
    Boolean[] values
)
```

setSSLStrictVerifyByLocation(location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setSSLStrictVerifyByLocation(
    String location
    String[] names
    Boolean[] values
)
```

setSSLSupportSSL2(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

This method is now deprecated.

```java
void setSSLSupportSSL2(
    String[] names
    Pool.SSLSupportSSL2[] values
)
```
setSSLSupportSSL2ByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

This method is now deprecated. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportSSL2ByLocation(
    String location
    String[] names
    Pool.SSLSupportSSL2[] values
)
```

setSSLSupportSSL3( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether SSLv3 is enabled for this pool

```java
void setSSLSupportSSL3(
    String[] names
    Pool.SSLSupportSSL3[] values
)
```

setSSLSupportSSL3ByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether SSLv3 is enabled for this pool This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportSSL3ByLocation(
    String location
    String[] names
    Pool.SSLSupportSSL3[] values
)
```

setSSLSupportTLS1( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether TLSv1.0 is enabled for this pool

```java
void setSSLSupportTLS1(
    String[] names
    Pool.SSLSupportTLS1[] values
)
```

setSSLSupportTLS11( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether TLSv1.1 is enabled for this pool

```java
void setSSLSupportTLS11(
    String[] names
    Pool.SSLSupportTLS11[] values
)
```

setSSLSupportTLS11ByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether TLSv1.1 is enabled for this pool This is a location specific function, any action will operate on the specified location.
Pool

void setSSLSupportTLS11ByLocation(  
    String location  
    String[] names
    Pool.SSLSupportTLS11[] values
)

setSSLSupportTLS12( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set whether TLSv1.2 is enabled for this pool

void setSSLSupportTLS12(  
    String[] names
    Pool.SSLSupportTLS12[] values
)

setSSLSupportTLS12ByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set whether TLSv1.2 is enabled for this pool This is a location specific function, any action will operate on the specified location.

void setSSLSupportTLS12ByLocation(  
    String location
    String[] names
    Pool.SSLSupportTLS12[] values
)

setSSLSupportTLS1ByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set whether TLSv1.0 is enabled for this pool This is a location specific function, any action will operate on the specified location.

void setSSLSupportTLS1ByLocation(  
    String location
    String[] names
    Pool.SSLSupportTLS1[] 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
)

setTransparentByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void setTransparentByLocation(  
    String location
    String[] names
    Boolean[] values
)
setUDPAcceptFrom( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set what sets of IP addresses and ports from which we should accept UDP responses.

```java
void setUDPAcceptFrom(
    String[] names
    Pool.UDPAcceptFrom[] values
)
```

setUDPAcceptFromByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set what sets of IP addresses and ports from which we should accept UDP responses. This is a location specific function, any action will operate on the specified location.

```java
void setUDPAcceptFromByLocation(
    String location
    String[] names
    Pool.UDPAcceptFrom[] values
)
```

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

Set the mask to validate the IP of UDP responses with. Only used if UDPAcceptFromIP is set to 'ip_mask'.

```java
void setUDPAcceptFromIPMask(
    String[] names
    String[] values
)
```

setUDPAcceptFromIPMaskByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the mask to validate the IP of UDP responses with. Only used if UDPAcceptFromIP is set to 'ip_mask'. This is a location specific function, any action will operate on the specified location.

```java
void setUDPAcceptFromIPMaskByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set the maximum delay in replying before a node receiving UDP packets is assumed to have failed.

```java
void setUDPResponseTimeout(
    String[] names
    Unsigned Integer[] values
)
```
setUDPResponseTimeoutByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum delay in replying before a node receiving UDP packets is assumed to have failed. This is a location specific function, any action will operate on the specified location.

void setUDPResponseTimeoutByLocation(String location, String[] names, Unsigned Integer[] values)

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;
}

Enumerations

Pool.AutoscaleIpstouse

enum Pool.AutoscaleIpstouse {
    # Public IP addresses
    publicips,

    # Private IP addresses
    privateips
}

Pool.LoadBalancingAlgorithm

enum Pool.LoadBalancingAlgorithm {
    # Round Robin
    roundrobin,
Function Reference

Pool

# Weighted Round Robin
wroundrobin,

# Perceptive
cells,

# Least Connections
connections,

# Weighted Least Connections
wconnections,

# Fastest Response Time
responsetimes,

# Random Node
random
}

Pool.NodeDeleteBehavior

enum Pool.NodeDeleteBehavior {
  # All connections to the node are closed immediately.
  immediate,

  # Allow existing connections to the node to finish before deletion.
  drain
}

Pool.SSLSupportSSL2

enum Pool.SSLSupportSSL2 {
  # Use the global setting for SSLv2
  use_default,

  # Enable SSLv2 (not recommended)
  enabled,

  # Disable SSLv2
  disabled
}

Pool.SSLSupportSSL3

enum Pool.SSLSupportSSL3 {
  # Use the global setting for SSLv3
  use_default,

  # Enable SSLv3
  enabled,

  # Disable SSLv3
  disabled
}

Pool.SSLSupportTLS1

enum Pool.SSLSupportTLS1 {
  # Use the global setting for TLSv1.0
  use_default,

  # Enable TLSv1.0

Pool.SSLSupportTLS11

enum Pool.SSLSupportTLS11 {
    # Use the global setting for TLSv1.1
    use_default,
    # Enable TLSv1.1
    enabled,
    # Disable TLSv1.1
    disabled
}

Pool.SSLSupportTLS12

enum Pool.SSLSupportTLS12 {
    # Use the global setting for TLSv1.2
    use_default,
    # Enable TLSv1.2
    enabled,
    # Disable TLSv1.2
    disabled
}

Pool.UDPAcceptFrom

enum Pool.UDPAcceptFrom {
    # Only the IP address and port to which the request was sent.
    dest_only,
    # Only the IP address to which the request was sent, but from any port.
    dest_ip_only,
    # Only a specific set of IP addresses, but from any port.
    ip_mask,
    # Any IP address and any port.
    all
}

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.
Methods

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

Add new back-end Traffic IP addresses to each of the named traffic IP groups.

```java
void addBackEndTrafficIPAddresses(
    String[] names
    String[][] values
)
```

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

Add new IP addresses to each of the named traffic IP groups.

```java
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.

```java
void addPassiveMachine(
    String[] names
    String[][] values
)
```

addTrafficIPGroup( names, details ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidInput, InvalidOperation

Add the new named Traffic IP Groups, using the provided details.

```java
void addTrafficIPGroup(
    String[] names
    TrafficIPGroups.Details[] details
)
```

addTrafficIPGroupWithMode( names, details ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidInput, InvalidOperation

Add the new named Traffic IP Groups, using the provided details.

```java
void addTrafficIPGroupWithMode(
    String[] names
    TrafficIPGroups.DetailsV2[] details
)
```

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

Add new traffic managers to each of the named traffic IP groups.

```java
void addTrafficManager(
    String[] names
)
deleteSpecificSubnetMappings( mappings ) throws InvalidInput

Delete specified interface network mappings.

```java
void deleteSpecificSubnetMappings(
    TrafficIPGroups.SubnetMappingPerHost[] mappings
)
```

deleteSubnetMappings()

Delete all interface network mappings.

```java
void deleteSubnetMappings()
```

deleteTrafficIPGroup( names ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, InvalidOperation

Delete the named Traffic IP Groups.

```java
void deleteTrafficIPGroup(
    String[] names
)
```

getAvailableTrafficManagers()

Get the names of all of the Traffic Managers in the cluster.

```java
String[] getAvailableTrafficManagers()
```

getBackEndTrafficIPAddresses( names ) throws ObjectDoesNotExist

Get IP addresses associated with the Traffic IP group that can be used for communication with back-end servers.

```java
String[][] getBackEndTrafficIPAddresses(
    String[] names
)
```

getEnabled( names ) throws ObjectDoesNotExist

Get whether this traffic IP group is enabled or not.

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

getIPAddresses( names ) throws ObjectDoesNotExist

Get the IP addresses that are managed by each of the named traffic IP groups.

```java
String[][] getIPAddresses(
    String[] names
)
```

getIPAssignmentMode( names ) throws ObjectDoesNotExist

Get the traffic IP distribution mode.
Function Reference

getIPAssignmentMode
getIPAssignmentMode

getIPDistributionMode
getIPDistributionMode

getKeepTogether
getKeepTogether

getMulticastIP
getMulticastIP

getNetworkInterfaces
getNetworkInterfaces

getNote
getNote

getPassiveMachine
getPassiveMachine

getRhiBgpMetricBase
getRhiBgpMetricBase

get how traffic IPs will be distributed across the machines in the cluster. If “multihosted” mode is used, the multicast IP must be set first.

Get the KeepTogether attribute for each of the named traffic IP groups.

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 'multihosted' distribution mode.

Get a list of network interfaces for all machines in the cluster.

Get the note for each of the named traffic IP groups.

Get the lists of passive machines in each of the named traffic IP groups.

Get the base BGP routing metric for this Traffic IP group.
getRhiBgpPassiveMetricOffset( names ) throws ObjectDoesNotExist
Get the BGP routing metric offset between active and passive traffic managers in this Traffic IP group.

Unsigned Integer[] getRhiBgpPassiveMetricOffset(
    String[] names
)

getRhiOspfv2MetricBase( names ) throws ObjectDoesNotExist
Get the base OSPFv2 routing metric for this Traffic IP group.

Unsigned Integer[] getRhiOspfv2MetricBase(
    String[] names
)

getRhiOspfv2PassiveMetricOffset( names ) throws ObjectDoesNotExist
Get the OSPFv2 routing metric offset between active and passive traffic managers in this Traffic IP group.

Unsigned Integer[] getRhiOspfv2PassiveMetricOffset(
    String[] names
)

getRhiProtocols( names ) throws ObjectDoesNotExist
Get the RHI protocols which may be used to advertise the addresses in the Traffic IP group.

String[] getRhiProtocols(
    String[] names
)

getSubnetMappings( hostnames ) throws InvalidInput
Get interface to CIDR subnet mappings.

TrafficIPGroups.GetSubnetMappingPerHost[] getSubnetMappings(
    String[] hostnames
)

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
)
Function Reference

TrafficIPGroups

removeBackEndTrafficIPAddresses( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation
Remove the named back-end Traffic IP addresses from each of the named traffic IP groups.

```java
void removeBackEndTrafficIPAddresses(
    String[] names
    String[][] values
)
```

removeIPAddresses( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation
Remove the named IP addresses from each of the named traffic IP groups.

```java
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.

```java
void removePassiveMachine(
    String[] names
    String[][] values
)
```

removeTrafficManager( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidOperation
Remove the named traffic managers from each named traffic IP group.

```java
void removeTrafficManager(
    String[] names
    String[][] values
)
```

renameTrafficIPGroup( names, new_names ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation
Rename each of the named Traffic IP Groups.

```java
void renameTrafficIPGroup(
    String[] names
    String[] new_names
)
```

setBackEndTrafficIPAddresses( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation
Set IP addresses associated with the Traffic IP group that can be used for communication with back-end servers.

```java
void setBackEndTrafficIPAddresses(
    String[] names
    String[][] values
)
```
setEnabled( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether this traffic IP group is enabled or not.

```java
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.

```java
void setIPAddresses(
    String[] names
    String[][] values
)
```

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

Set the traffic IP distribution mode.

```java
void setIPAssignmentMode(
    String[] names
    TrafficIPGroups.IPAssignmentMode[] values
)
```

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

Set how traffic IPs will be distributed across the machines in the cluster. If "multihosted" mode is used, the multicast IP must be set first.

```java
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.

```java
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 'multihosted' distribution mode.

```java
void setMulticastIP(
    String[] names
    String[] values
)
```
Function Reference

TrafficIPGroups

```
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
)

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

Set the base BGP routing metric for this Traffic IP group.

void setRhiBgpMetricBase(
    String[] names
    Unsigned Integer[] values
)

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

Set the BGP routing metric offset between active and passive traffic managers in this Traffic IP group.

void setRhiBgpPassiveMetricOffset(
    String[] names
    Unsigned Integer[] values
)

setRhiOspfv2MetricBase( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the base OSPFv2 routing metric for this Traffic IP group.

void setRhiOspfv2MetricBase(
    String[] names
    Unsigned Integer[] values
)

setRhiOspfv2PassiveMetricOffset( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the OSPFv2 routing metric offset between active and passive traffic managers in this Traffic IP group.

void setRhiOspfv2PassiveMetricOffset(
    String[] names
    Unsigned Integer[] values
)
```
setRhiProtocols( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the RHI protocols which may be used to advertise the addresses in the Traffic IP group.

```java
void setRhiProtocols(
    String[] names
    String[] values
)
```

setSubnetMappings( mappings ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Update interface to CIDR subnet mappings. Note: this function replaces your existing TrafficIP network mappings rather than adding to them. To avoid deleting any existing entries, you must include them in the specified update list.

```java
void setSubnetMappings(
    TrafficIPGroups.SubnetMappingPerHost[] mappings
)
```

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.

```java
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.

```java
void setUseClientSourcePort(
    String[] names
    Boolean[] values
)
```

**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.

```java
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;
}
```
TrafficIPGroups::DetailsV2

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.

```c
struct TrafficIPGroups::DetailsV2 {
    # The IP addresses in the Traffic IP Group.
    String[] ipaddresses;

    # The IP distribution mode of the traffic IP group.
    TrafficIPGroups::IPDistributionMode mode;

    # The multicast IP address of a multihosted traffic IP group.
    String multicast;

    # The names of the traffic managers that will manage the IP Addresses.
    String[] machines;

    # The names of the traffic managers that will be passive / standby members of
    # the group.
    String[] passive_machines;
}
```

TrafficIPGroups::NetworkInterface

This structure displays the network interfaces of all machines in the cluster.

```c
struct TrafficIPGroups::NetworkInterface {
    # The traffic manager in the cluster.
    String hostname;

    # The network interfaces configured in this traffic manager.
    String[] interfaces;
}
```

TrafficIPGroups::SubnetMapping

This structure contains mappings of network interface to CIDR subnets. These mappings are used to raise a TrafficIP on a desired interface.

```c
struct TrafficIPGroups::SubnetMapping {
    # The interface on the system.
    String interface;

    # The subnets mappings for the interface.
    String[] subnets;
}
```

TrafficIPGroups::SubnetMappingPerHost

This structure shows the traffic IP subnet mapping per host machine in the cluster.

```c
struct TrafficIPGroups::SubnetMappingPerHost {
    # The traffic manager in the cluster.
    String hostname;

    # The subnets mappings for this traffic manager.
    TrafficIPGroups::SubnetMapping[] subnetmappings;
}
```
Enumerations

TrafficIPGroups.IPAssignmentMode

```java
enum TrafficIPGroups.IPAssignmentMode {
    # Alphabetical order of traffic manager hostnames
    alphabetic,

    # Approximately balanced between traffic managers
    balanced
}
```

TrafficIPGroups.IPDistributionMode

```java
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 route health injection to route traffic to the active machine - IPv4
    # only
    rhi,

    # Use an EC2-Classic Elastic IP address.
    ec2elastic,

    # Use an EC2-VPC Elastic IP address.
    ec2vpcelastic,

    # Use an EC2-VPC Private IP address.
    ec2vpcprivate
}
```

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.

Methods

```java
addRule( names, texts ) throws InvalidObjectName, ObjectAlreadyExists, DeploymentError
```

Add new rules to the catalog.

```java
void addRule{
    String[] names
    String[] texts
}
```
Function Reference

Catalog.Rule

**checkSyntax( rule_text )**
Check the syntax of each of the supplied TrafficScript rule texts. This method does not modify any configuration.

```java
Catalog.Rule.SyntaxCheck[] checkSyntax(
    String[] rule_text
)
```

**copyRule( names, new_names ) throws InvalidObjectName, ObjectAlreadyExists, ObjectDoesNotExist, DeploymentError**
Copy the named rules in the catalog.

```java
void copyRule(
    String[] names
    String[] new_names
)
```

**deleteRule( names ) throws ObjectInUse, ObjectDoesNotExist, DeploymentError, InvalidOperation**
Delete the named rules from the catalog.

```java
void deleteRule(
    String[] names
)
```

**getRuleDetails( names ) throws ObjectDoesNotExist, DeploymentError**
Get the rule text and notes (if any), for each of the named rules.

```java
Catalog.Rule.RuleInfo[] getRuleDetails(
    String[] names
)
```

**getRuleNames()**
Get the names of all rules in the catalog.

```java
String[] getRuleNames()
```

**renameRule( names, new_names ) throws InvalidObjectName, ObjectAlreadyExists, ObjectDoesNotExist, DeploymentError, InvalidOperation**
Rename the named rules in the catalog.

```java
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.

```java
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.

```java
void setRuleText(
    String[] names
    String[] text
)
```

**Structures**

**Catalog.Rule.RuleInfo**

This structure contains basic information for a rule in the catalog.

```java
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.

```java
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;
}
```

**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.

**Methods**

**addMonitors( names ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError**

Add new monitors (defaults to TCP transaction monitor, monitoring each node separately).

```java
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`.

```java
void addProgramArguments(
    String[] names
    Catalog.Monitor.Argument[][] arguments
)
```

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

Copy the named monitors.

```java
void copyMonitors(
    String[] names
    String[] new_names
)
```

deleteMonitorProgram( names ) throws ObjectDoesNotExist, DeploymentError, ObjectInUse

Delete the named monitor programs.

```java
void deleteMonitorProgram(
    String[] names
)
```

deleteMonitors( names ) throws ObjectDoesNotExist, InvalidOperation, DeploymentError, ObjectInUse

Delete these monitors.

```java
void deleteMonitors(
    String[] names
)
```

downloadMonitorProgram( name ) throws ObjectDoesNotExist

Download the named monitor program.

```java
Binary Data downloadMonitorProgram(
    String name
)
```

getAllMonitorNames()

Get the names of all monitors.

```java
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.

```java
String[] getAuthentication(
    String[] names
)
getAuthenticationByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the authentication (user:password) that each of the named monitors should use in the test HTTP request. This is a location specific function, any action will operate on the specified location.

```java
String[] getAuthenticationByLocation(
    String location
    String[] names
)
```

getBackOff( names ) throws ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors should back-off after it has failed.

```java
Boolean[] getBackOff(
    String[] names
)
```

getBackOffByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors should back-off after it has failed. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getBackOffByLocation(
    String location
    String[] names
)
```

getBodyRegex( names ) throws ObjectDoesNotExist, InvalidOperation

Get the body regular expression that that each of the named monitors' HTTP response must match.

```java
String[] getBodyRegex(
    String[] names
)
```

getBodyRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the body regular expression that that each of the named monitors' HTTP response must match. This is a location specific function, any action will operate on the specified location.

```java
String[] getBodyRegexByLocation(
    String location
    String[] names
)
```

getcCloseString( names ) throws ObjectDoesNotExist, InvalidOperation

Get an optional string that each of the named monitors should write to the server before closing the connection.

```java
String[] getCloseString(
    String[] names
)
```
getCloseStringByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get an optional string that each of the named monitors should write to the server before closing the connection. This is a location specific function, any action will operate on the specified location.

```java
String[] getCloseStringByLocation(
    String location
    String[] names
)
```

getCustomMonitorNames()

Get the names of all the custom monitors.

```java
String[] getCustomMonitorNames()
```

ggetDelay( names ) throws ObjectDoesNotExist, InvalidOperation

Get the minimum time between calls to each of the named monitors (in seconds).

```java
Unsigned Integer[] getDelay(
    String[] names
)
```

ggetDelayByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the minimum time between calls to each of the named monitors (in seconds). This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getDelayByLocation(
    String location
    String[] names
)
```

ggetFailures( names ) throws ObjectDoesNotExist, InvalidOperation

Get the number of failures required, by each of the named monitors, before a node is classed as unavailable.

```java
Unsigned Integer[] getFailures(
    String[] names
)
```

ggetFailuresByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the number of failures required, by each of the named monitors, before a node is classed as unavailable. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getFailuresByLocation(
    String location
    String[] names
)
```

ggetHealthOnly( names ) throws ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors should monitor health only (ignore load).

```java
Boolean[] getHealthOnly(
```
getHealthOnlyByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors should monitor health only (ignore load). This is a location specific function, any action will operate on the specified location.

Boolean[] getHealthOnlyByLocation(
    String location
    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
)

getHostHeaderByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the host header that each of the named monitors should use in the test HTTP request. This is a location specific function, any action will operate on the specified location.

String[] getHostHeaderByLocation(
    String location
    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
)

g MACHINE ByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

String[] getMachineByLocation(
    String location
    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
)
Function Reference

Catalog: Monitor

String[] names

getMaxResponseLenByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxResponseLenByLocation(
    String location
    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
)

getPathByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the path that each of the named monitors should use in the test HTTP request. This is a location specific function, any action will operate on the specified location.

String[] getPathByLocation(
    String location
    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

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
)

getResponseRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation
Get the regular expression that each of the named monitors should match against the server response. This is a location specific function, any action will operate on the specified location.

String[] getResponseRegexByLocation(
    String location
    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
)

getRtspBodyRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation
Get the body regular expression that each of the named monitors' RTSP response must match. This is a location specific function, any action will operate on the specified location.

String[] getRtspBodyRegexByLocation(
    String location
    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
)

getRtspPathByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation
Get the path that each of the named monitors should use in the test RTSP request. This is a location specific function, any action will operate on the specified location.

String[] getRtspPathByLocation(
    String location
    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
)

getRtspStatusRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the status code regular expression that each of the named monitors’ RTSP response must match. This is a location specific function, any action will operate on the specified location.

String[] getRtspStatusRegexByLocation(
    String location
    String[] names
)

getScope( names ) throws ObjectDoesNotExist, InvalidOperation

Get the scope of each named monitor.

Catalog.Monitor.Scope[] getScope(
    String[] names
)

g 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
)

g getSipBodyRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the body regular expression that that each of the named monitors’ SIP response must match. This is a location specific function, any action will operate on the specified location.

String[] getSipBodyRegexByLocation(
    String location
    String[] names
)

g 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
)

g getSipStatusRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation

Get the status code regular expression that that each of the named monitors’ SIP response must match. This is a location specific function, any action will operate on the specified location.
String[] getSipStatusRegexByLocation(
    String location
    String[] names
)

getSipTransport( names ) throws ObjectDoesNotExist, InvalidOperation
Get the transport protocol that the monitor will use
Catalog.Monitor.SipTransport[] getSipTransport(
    String[] names
)

getSipTransportByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation
Get the transport protocol that the monitor will use This is a location specific function, any action will operate on the specified location.
Catalog.Monitor.SipTransport[] getSipTransportByLocation(
    String location
    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
)

getStatusRegexByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation
Get the status code regular expression that that each of the named monitors' HTTP response must match. This is a location specific function, any action will operate on the specified location.
String[] getStatusRegexByLocation(
    String location
    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
)

getTimeoutByLocation( location, 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). This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getTimeoutByLocation(
    String location
Function Reference

Catalog.Monitor

String[] names
)

**getTypes(names) throws** ObjectDoesNotExist, InvalidOperation

Get the internal monitor type to use for each named monitor.

Catalog.Monitor.Type[] getType(
    String[] names
)

**getUDPAcceptAll(names) throws** ObjectDoesNotExist, InvalidOperation

Get if the monitor should accept responses from any IP and port. UDP monitors only.

Boolean[] getUDPAcceptAll(
    String[] names
)

**getUDPAcceptAllByLocation(location, names) throws** ObjectDoesNotExist,
InvalidOperation

Get if the monitor should accept responses from any IP and port. UDP monitors only. This is a location
specific function, any action will operate on the specified location.

Boolean[] getUDPAcceptAllByLocation(
    String location
    String[] names
)

**getUseSSL(names) throws** ObjectDoesNotExist, InvalidOperation

Get whether each of the named monitors can connect using SSL.

Boolean[] getUseSSL(
    String[] names
)

**getUseSSLByLocation(location, names) throws** ObjectDoesNotExist,
InvalidOperation

Get whether each of the named monitors can connect using SSL. This is a location
specific function, any action will operate on the specified location.

Boolean[] getUseSSLByLocation(
    String location
    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
)
**getVerboseByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation**

Get whether each of the named monitors should emit verbose logging (useful for diagnostics). This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getVerboseByLocation(
    String location
    String[] names
)
```

**getWriteString( names ) throws ObjectDoesNotExist, InvalidOperation**

Get the string that each of the named monitors should write down the TCP connection.

```java
String[] getWriteString(
    String[] names
)
```

**getWriteStringByLocation( location, names ) throws ObjectDoesNotExist, InvalidOperation**

Get the string that each of the named monitors should write down the TCP connection. This is a location specific function, any action will operate on the specified location.

```java
String[] getWriteStringByLocation(
    String location
    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'.

```java
void removeProgramArguments(
    String[] names
    String[][] arguments
)
```

**renameMonitors( names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation**

Rename these monitors.

```java
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.

```java
void setAuthentication(
    String[] names
    String[] values
)
```
setAuthenticationByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void setAuthenticationByLocation(
    String location
    String[] names
    String[] values
)

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

Set whether each of the named monitors should back-off after it has failed.

void setBackOff(
    String[] names
    Boolean[] values
)

setBackOffByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set whether each of the named monitors should back-off after it has failed. This is a location specific function, any action will operate on the specified location.

void setBackOffByLocation(
    String location
    String[] names
    Boolean[] 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
)

setBodyRegexByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the body regular expression that that each of the named monitors' HTTP response must match. This is a location specific function, any action will operate on the specified location.

void setBodyRegexByLocation(
    String location
    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.

```java
void setCloseString{
    String[] names
    String[] values
}
```

setCloseStringByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setCloseStringByLocation{
    String location
    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).

```java
void setDelay{
    String[] names
    Unsigned Integer[] values
}
```

setDelayByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the minimum time between calls to each of the named monitors (in seconds). This is a location specific function, any action will operate on the specified location.

```java
void setDelayByLocation{
    String location
    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.

```java
void setFailures{
    String[] names
    Unsigned Integer[] values
}
```
**setFailuresByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setFailuresByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```

**setHealthOnly( names, values )** throws **ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation**

Set whether each of the named monitors should monitor health only (ignore load).

```java
void setHealthOnly(
    String[] names
    Boolean[] values
)
```

**setHealthOnlyByLocation( location, names, values )** throws **ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation**

Set whether each of the named monitors should monitor health only (ignore load). This is a location specific function, any action will operate on the specified location.

```java
void setHealthOnlyByLocation(
    String location
    String[] names
    Boolean[] 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.

```java
void setHostHeader(
    String[] names
    String[] values
)
```

**setHostHeaderByLocation( location, names, values )** throws **ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation**

Set the host header that each of the named monitors should use in the test HTTP request. This is a location specific function, any action will operate on the specified location.

```java
void setHostHeaderByLocation(
    String location
    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
)

setMachineByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

void setMachineByLocation(
    String location
    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
)

setMaxResponseLenByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

void setMaxResponseLenByLocation(
    String location
    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.

```java
void setPath(
    String[] names
    String[] values
)
```

setPathByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the path that each of the named monitors should use in the test HTTP request. This is a location specific function, any action will operate on the specified location.

```java
void setPathByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set the name of the program that each named monitor runs.

```java
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.

```java
void setResponseRegex(
    String[] names
    String[] values
)
```

setResponseRegexByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the regular expression that each of the named monitors should match against the server response. This is a location specific function, any action will operate on the specified location.

```java
void setResponseRegexByLocation(
    String location
    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.

```java
void setRtspBodyRegex(
```
String[] names
String[] values
)

setRtspBodyRegexByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the body regular expression that each of the named monitors’ RTSP response must match. This is a location specific function, any action will operate on the specified location.

void setRtspBodyRegexByLocation(  
    String location  
    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  
)

setRtspPathByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the path that each of the named monitors should use in the test RTSP request. This is a location specific function, any action will operate on the specified location.

void setRtspPathByLocation(  
    String location  
    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  
)

setRtspStatusRegexByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the status code regular expression that each of the named monitors’ RTSP response must match. This is a location specific function, any action will operate on the specified location.

void setRtspStatusRegexByLocation(  
    String location  
    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 each of the named monitors’ SIP response must match.

void setSipBodyRegex{
    String[] names
    String[] values
}

setSipBodyRegexByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the body regular expression that each of the named monitors’ SIP response must match. This is a location specific function, any action will operate on the specified location.

void setSipBodyRegexByLocation{
    String location
    String[] names
    String[] values
}

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

Set the status code regular expression that each of the named monitors’ SIP response must match.

void setSipStatusRegex{
    String[] names
    String[] values
}

setSipStatusRegexByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the status code regular expression that each of the named monitors’ SIP response must match. This is a location specific function, any action will operate on the specified location.

void setSipStatusRegexByLocation{
    String location
    String[] names
    String[] values
}

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

Set the transport protocol that the monitor will use.

void setSipTransport{

setSipTransportByLocation(location, names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the transport protocol that the monitor will use. This is a location specific function, any action will operate on the specified location.

void setSipTransportByLocation(
    String location,
    String[] names,
    Catalog.Monitor.SipTransport[] values
)

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

Set the status code regular expression that each of the named monitors’ HTTP response must match.

void setStatusRegex(
    String[] names,
    String[] values
)

setStatusRegexByLocation(location, names, values) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the status code regular expression that each of the named monitors’ HTTP response must match. This is a location specific function, any action will operate on the specified location.

void setStatusRegexByLocation(
    String location,
    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
)

setTimeoutByLocation(location, 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). This is a location specific function, any action will operate on the specified location.

void setTimeoutByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
**setType(names, values)** throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the internal monitor type to use for each named monitor.

```java
void setType(
    String[] names
    Catalog.Monitor.Type[] values
)
```

**setUDPAcceptAll(names, values)** throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set if the monitor should accept responses from any IP and port. UDP monitors only.

```java
void setUDPAcceptAll(
    String[] names
    Boolean[] values
)
```

**setUDPAcceptAllByLocation(location, names, values)** throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set if the monitor should accept responses from any IP and port. UDP monitors only. This is a location specific function, any action will operate on the specified location.

```java
void setUDPAcceptAllByLocation(
    String location
    String[] names
    Boolean[] values
)
```

**setUseSSL(names, values)** throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set whether each of the named monitors can connect using SSL.

```java
void setUseSSL(
    String[] names
    Boolean[] values
)
```

**setUseSSLByLocation(location, names, values)** throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set whether each of the named monitors can connect using SSL. This is a location specific function, any action will operate on the specified location.

```java
void setUseSSLByLocation(
    String location
    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).

```java
void setVerbose(
```
setVerboseByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set whether each of the named monitors should emit verbose logging (useful for diagnostics). This is a location specific function, any action will operate on the specified location.

```java
void setVerboseByLocation(
    String location
    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.

```java
void setWriteString(
    String[] names
    String[] values
)
```

setWriteStringByLocation( location, names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Set the string that each of the named monitors should write down the TCP connection. This is a location specific function, any action will operate on the specified location.

```java
void setWriteStringByLocation(
    String location
    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'.

```java
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.

```java
void uploadMonitorProgram(
    String name
    Binary Data content
)
Structures

Catalog.Monitor.Argument
An argument that is added to the command line when the monitor program is run

```c
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;
}
```

Enumerations

Catalog.Monitor.Scope
```
enum Catalog.Monitor.Scope {
    # Node: Monitor each node in the pool separately
    pernode,
    
    # Pool/GLB: Monitor a specified machine
    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,
```
Catalog.SSL.Certificates

URI: http://soap.zeus.com/zxtm/1.1/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.

**Methods**

**createSelfSignedCertificate( names, details )** throws InvalidObjectName, ObjectAlreadyExists, InvalidInput, DeploymentError

Create new self-signed certificates.

```java
void createSelfSignedCertificate(
    String[] names
    Catalog.SSL.Certificates.CertificateDetails[] details
)
```

**createSelfSignedECDSACertificate( names, details )** throws InvalidObjectName, ObjectAlreadyExists, InvalidInput, DeploymentError

Create new self-signed ECDSA certificates.

```java
void createSelfSignedECDSACertificate(
    String[] names
    Catalog.SSL.Certificates.ECCertificateDetails[] details
)
```

**deleteCertificate( names )** throws InvalidObjectName, ObjectDoesNotExist, InvalidOperation, DeploymentError

Delete the named certificates.

```java
void deleteCertificate(
    String[] names
)
```

**deleteCertificateHW( names )** throws InvalidObjectName, ObjectDoesNotExist, InvalidOperation, DeploymentError

Delete the named certificates even if they are stored on secure hardware and could be in use by other clients of the hardware.

```java
void deleteCertificateHW(
    String[] names
)
```
**getCertificateInfo( names ) throws ObjectDoesNotExist**

Get the information about the named certificates.

```java
Certificate[] getCertificateInfo(
    String[] names
)
```

**getCertificateNames()**

Get the names of the installed certificates.

```java
String[] getCertificateNames()
```

**getCertificateRequest( names ) throws ObjectDoesNotExist**

Get Certificate signing requests for the named certificates.

```java
String[] getCertificateRequest(
    String[] names
)
```

**getRawCertificate( names ) throws ObjectDoesNotExist**

Get the raw (PEM-encoded) certificates.

```java
String[] getRawCertificate(
    String[] names
)
```

**importCertificate( names, keys ) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput**

Create a new certificate, importing the certificate and private key.

```java
void importCertificate(
    String[] names
    CertificateFiles[] keys
)
```

**renameCertificate( names, new_names ) throws InvalidObjectName, ObjectAlreadyExists, ObjectDoesNotExist**

Rename the named certificates.

```java
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.

```java
void setRawCertificate(
    String[] names
    String[] certs
)
```
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 valid for (e.g. 365 for 1
    # year validity)
    Integer valid_days;

    # The size of the generated private key. Use 2048 for normal use, or 3072 for
    # more security
    Integer key_size;
}
```

Catalog.SSL.Certificates.ECCertificateDetails

This structure contains the information used when generating self-signed test certificates with ECDSA keys.

```
struct Catalog.SSL.Certificates.ECCertificateDetails {
    # 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 valid for (e.g. 365 for 1
    # year validity)
    Integer valid_days;

    # The name of the curve used to generate the private key. Possible values
    # include P256, P384 and P521 in order of increasing strength.
    String curve;
}
```

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 type of key used in the certificate.
    String key_type;
}
```
# The DSA public key 'y' used in the certificate.
String y;

# The first coordinate of the public key Q for an ECDSA key used in the
# certificate.
String Q_x;

# The second coordinate of the public key Q for an ECDSA key used in the
# certificate.
String Q_y;

# The name of the curve used by the ECDSA key used in the certificate.
String curve;

# The RSA modulus of the certificate.
String modulus;

# The RSA 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 organization
    String organization;
}
## Catalog.SSL.CertificateAuthorities

URI: `http://soap.zeus.com/zxtm/1.1/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.

### Methods

#### deleteCertificateAuthority( names ) throws ObjectDoesNotExist

Delete the named Certificate Authority and associated Revocation list.

```java
void deleteCertificateAuthority(
    String[] names
)
```

#### getCertificateAuthorityInfo( names ) throws ObjectDoesNotExist

Get the Certificate Information, and the revoked certificates.

```java
Catalog.SSL.CertificateAuthorities.Details[] getCertificateAuthorityInfo(
    String[] names
)
```

#### getCertificateAuthorityNames()

Get the names of the configured Certificate Authorities.

```java
String[] getCertificateAuthorityNames()
```

#### getRawCertificate( names ) throws ObjectDoesNotExist

Get the raw PEM encoded Certificate for the named Certificate Authorities.

```java
String[] getRawCertificate(
    String[] names
)
```

#### importCRL( crls ) throws InvalidInput, ObjectDoesNotExist

Import Certificate Revocation Lists. The associated Certificate Authority certificates should already be imported.

```java
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
)

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.

```c
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 type of key used in the certificate.
    String key_type;

    # The DSA public key 'y' used in the certificate.
    String y;

    # The first coordinate of the public key Q for an ECDSA key used in the certificate.
    String Q_x;

    # The second coordinate of the public key Q for an ECDSA key used in the certificate.
    String Q_y;

    # The name of the curve used by the ECDSA key used in the certificate.
    String curve;

    # The RSA modulus of the certificate.
    String modulus;

    # The RSA 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.

```c
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 organization
    String organization;

    # The unit inside the organization
    String unit;

    # An email address. This is usually empty.
    String email;
}
```

---

## Catalog.SSL.ClientCertificates

**URI:** http://soap.zeus.com/zxtm/1.1/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.

### Methods

**deleteClientCertificate( names ) throws ObjectDoesNotExist, InvalidOperation, DeploymentError**

Delete the named client certificates.

```java
void deleteClientCertificate(
    String[] names
)
```
deleteClientCertificateHW( names ) throws ObjectDoesNotExist, InvalidOperation, DeploymentError

Delete the named client certificates even if they are stored on secure hardware and could be in use by other clients of the hardware.

```java
void deleteClientCertificateHW(
    String[] names
)
```

getClientCertificateInfo( names ) throws ObjectDoesNotExist

Get information about the named client certificates.

```java
Certificate[] getClientCertificateInfo(
    String[] names
)
```

getClientCertificateNames()

Get the names of the installed client certificates.

```java
String[] getClientCertificateNames()
```

importClientCertificate( names, keys ) throws InvalidObjectName, ObjectAlreadyExists, InvalidInput

Import client certificates and associated private keys.

```java
void importClientCertificate(
    String[] names
    CertificateFiles[] keys
)
```

renameClientCertificate( names, new_names ) throws ObjectAlreadyExists, ObjectDoesNotExist, DeploymentError

Rename the named client certificates.

```java
void renameClientCertificate(
    String[] names
    String[] new_names
)
```

Structures

Certificate

This structure contains information (such as the subject and issuer) about a certificate.

```java
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 type of key used in the certificate.
String key_type;

# The DSA public key 'y' used in the certificate.
String y;

# The first coordinate of the public key Q for an ECDSA key used in the certificate.
String Q_x;

# The second coordinate of the public key Q for an ECDSA key used in the certificate.
String Q_y;

# The name of the curve used by the ECDSA key used in the certificate.
String curve;

# The RSA modulus of the certificate.
String modulus;

# The RSA 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 organization
String organization;

# The unit inside the organization
String unit;

# An email address. This is usually empty.
String email;

}}

---

**Catalog.SSL.DNSSEC**

URI: [http://soap.zeus.com/zxtm/1.1/Catalog/SSL/DNSSEC/](http://soap.zeus.com/zxtm/1.1/Catalog/SSL/DNSSEC/)

The Catalog.SSL.DNSSEC interface allows management of the DNSSEC private keys used to alter signed GLB DNS responses.

## Methods

### addKeysWithManualIDs( names, keys, ids, algs ) throws ObjectAlreadyExists, InvalidObjectName, InvalidInput

Upload a DNSSEC private key to the traffic manager's catalog. Each key string should be the contents of your private key file. The ID of the key is the third set of parameters. The fourth parameter gives the IANA DNSSEC algorithm number for each key, for example RSA/SHA-1 is 5.

```java
void addKeysWithManualIDs(
    String[] names
    String[] keys
    Integer[] ids
    Integer[] algs
)
```

### addStandardKeys( names, keys ) throws ObjectAlreadyExists, InvalidObjectName, InvalidInput

Upload a DNSSEC private key to the traffic manager's catalog. Each key string should be the contents of your private key file. The ID of the key and the algorithm will be calculated assuming it is a standard ZSK.

```java
void addStandardKeys(
    String[] names
    String[] keys
)
```

### deleteKeys( names ) throws ObjectInUse, ObjectDoesNotExist, DeploymentError

Delete the specified DNSSEC keys.
void deleteKeys(
    String[] names
)

getKeyIDs( names ) throws ObjectDoesNotExist
Get the IDs of the specified DNSSEC private keys.

Integer[] getKeyIDs(
    String[] names
)

getKeyNames()
Get the names of the installed DNSSEC private keys.

String[] getKeyNames()

renameKeys( names, new_names ) throws ObjectDoesNotExist, InvalidInput, InvalidObjectName, ObjectAlreadyExists, DeploymentError
Rename the specified DNSSEC keys.

void renameKeys(
    String[] names
    String[] new_names
)

---

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.

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
}

addAllowedAddressesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void addAllowedAddressesByLocation{
    String location
}
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.

```java
void addBannedAddresses(
    String[] class_names
    String[][] values
)
```

addBannedAddressesByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void addBannedAddressesByLocation(
    String location
    String[] class_names
    String[][] values
)
```

addProtection( class_names ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add new Protection classes.

```java
void addProtection(
    String[] class_names
)
```

copyProtection( class_names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Copy the named Protection classes.

```java
void copyProtection(
    String[] class_names
    String[] new_names
)
```

deleteProtection( class_names ) throws ObjectInUse, ObjectDoesNotExist, DeploymentError

Delete the named Protection classes.

```java
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.

```java
String[][] getAllowedAddresses(
    String[] class_names
)
```
getAllowedAddressesByLocation( location, class_names ) throws ObjectDoesNotExist

Get the list of IP addresses and CIDR IP subnets that are always allowed access. This is a location specific function, any action will operate on the specified location.

String[][] getAllowedAddressesByLocation(
    String location
    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
)

getBannedAddressesByLocation( location, class_names ) throws ObjectDoesNotExist

Get the list of IP addresses and CIDR IP subnets that aren't allowed access. This is a location specific function, any action will operate on the specified location.

String[][] getBannedAddressesByLocation(
    String location
    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
)

getDebugByLocation( location, class_names ) throws ObjectDoesNotExist

Get whether the service protection classes are in debug mode. When in debug mode, verbose log messages are written. This is a location specific function, any action will operate on the specified location.

Boolean[] getDebugByLocation(
    String location
    String[] class_names
)

getEnabled( class_names ) throws ObjectDoesNotExist

Get whether the service protection classes are enabled.

Boolean[] getEnabled(
    String[] class_names
)
getEnabledByLocation( location, class_names ) throws ObjectDoesNotExist
Get whether the service protection classes are enabled. This is a location specific function, any action will operate on the specified location.

Boolean[] getEnabledByLocation(
    String location
    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
)

getHTTPCheckRfc2396ByLocation( location, class_names ) throws ObjectDoesNotExist
Get whether requests with poorly-formed URLs (as specified in RFC 2396) should be rejected. This is a location specific function, any action will operate on the specified location.

Boolean[] getHTTPCheckRfc2396ByLocation(
    String location
    String[] class_names
)

getHTTPRejectBinary( class_names ) throws ObjectDoesNotExist
Get whether requests containing binary data (after decoding) should be rejected.

Boolean[] getHTTPRejectBinary(
    String[] class_names
)

getHTTPRejectBinaryByLocation( location, class_names ) throws ObjectDoesNotExist
Get whether requests containing binary data (after decoding) should be rejected. This is a location specific function, any action will operate on the specified location.

Boolean[] getHTTPRejectBinaryByLocation(
    String location
    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
)
getHTTPSendErrorPageByLocation( location, class_names ) throws ObjectDoesNotExist

Get whether an HTTP error message should be sent when a connection is dropped, rather than just dropping the connection. This is a location specific function, any action will operate on the specified location.

Boolean[] getHTTPSendErrorPageByLocation(
    String location
    String[] class_names
)

getLogInterval( class_names ) throws ObjectDoesNotExist

Get the interval between logging service protection messages (in seconds).

Unsigned Integer[] getLogInterval(
    String[] class_names
)

getLogIntervalByLocation( location, class_names ) throws ObjectDoesNotExist

Get the interval between logging service protection messages (in seconds). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getLogIntervalByLocation(
    String location
    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
)

getMax10ConnectionsByLocation( location, class_names ) throws ObjectDoesNotExist

Get the maximum number of simultaneous connections allowed from the 10 busiest IP addresses. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMax10ConnectionsByLocation(
    String location
    String[] class_names
)

getMax1Connections( class_names ) throws ObjectDoesNotExist

Get the maximum number of simultaneous connections allowed from an individual IP address (0 means unlimited).

Unsigned Integer[] getMax1Connections(
    String[] class_names
)
getMax1ConnectionsByLocation( location, class_names ) throws ObjectDoesNotExist

Get the maximum number of simultaneous connections allowed from an individual IP address (0 means unlimited). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMax1ConnectionsByLocation(
    String location,
    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
)

getMaxConnectionRateByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxConnectionRateByLocation(
    String location,
    String[] class_names
)

getMaxHTTPBodyLength( class_names ) throws ObjectDoesNotExist

Get the maximum size of the HTTP request body data (in bytes, 0 means no limit).

Unsigned Integer[] getMaxHTTPBodyLength(
    String[] class_names
)

getMaxHTTPBodyLengthByLocation( location, class_names ) throws ObjectDoesNotExist

Get the maximum size of the HTTP request body data (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxHTTPBodyLengthByLocation(
    String location,
    String[] class_names
)

getMaxHTTPHeaderLength( class_names ) throws ObjectDoesNotExist

Get the maximum size of a single HTTP request header (in bytes, 0 means no limit).

Unsigned Integer[] getMaxHTTPHeaderLength(
    String[] class_names
)
getMaxHTTPHeaderLengthByLocation( location, class_names ) throws ObjectDoesNotExist

Get the maximum size of a single HTTP request header (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxHTTPHeaderLengthByLocation(  
    String location  
    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  
)

getMaxHTTPRequestLengthByLocation( location, class_names ) throws ObjectDoesNotExist

Get the maximum size of all the HTTP request headers (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxHTTPRequestLengthByLocation(  
    String location  
    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  
)

getMaxHTTPURLLengthByLocation( location, class_names ) throws ObjectDoesNotExist

Get the maximum size of the request URL (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaxHTTPURLLengthByLocation(  
    String location  
    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  
)
getMinConnectionsByLocation( location, class_names ) throws ObjectDoesNotExist

Get the number of simultaneous connections that are always allowed from each IP address (0 means unlimited). This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMinConnectionsByLocation(
    String location
    String[] class_names
)

gernote( class_names ) throws ObjectDoesNotExist

Get the note for each of the named Protection classes

String[] getNote(
    String[] class_names
)

getPerProcessConnectionCount( class_names ) throws ObjectDoesNotExist

Get whether or not each process within a Traffic Manager counts connections independently, when limiting the maximum simultaneous connections allowed from one IP address.

Boolean[] getPerProcessConnectionCount(
    String[] class_names
)

getPerProcessConnectionCountByLocation( location, class_names ) throws ObjectDoesNotExist

Get whether or not each process within a Traffic Manager counts connections independently, when limiting the maximum simultaneous connections allowed from one IP address. This is a location specific function, any action will operate on the specified location.

Boolean[] getPerProcessConnectionCountByLocation(
    String location
    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. The valid range is 1-99999 seconds.

Unsigned Integer[] getRateTimer(
    String[] class_names
)
getRateTimerByLocation( location, 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. The valid range is 1-99999 seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getRateTimerByLocation(
    String location
    String[] class_names
)

getRule( class_names ) throws ObjectDoesNotExist

Get the TrafficScript rule to be applied to all connections.

String[] getRule(
    String[] class_names
)

getRuleByLocation( location, class_names ) throws ObjectDoesNotExist

Get the TrafficScript rule to be applied to all connections. This is a location specific function, any action will operate on the specified location.

String[] getRuleByLocation(
    String location
    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
)

getTestingByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

Boolean[] getTestingByLocation(
    String location
    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
)
removeAllowedAddressesByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Remove IP addresses and CIDR IP subnets from the list of machines that are always allowed access. This is a location specific function, any action will operate on the specified location.

```java
void removeAllowedAddressesByLocation{
    String location
    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.

```java
void removeBannedAddresses{
    String[] class_names
    String[][] values
}
```

removeBannedAddressesByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Remove IP addresses and CIDR IP subnets from the list of machines that aren't allowed access. This is a location specific function, any action will operate on the specified location.

```java
void removeBannedAddressesByLocation{
    String location
    String[] class_names
    String[][] values
}
```

renameProtection( class_names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename the named Protection classes.

```java
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.

```java
void setAllowedAddresses{
    String[] class_names
    String[][] values
}
```

setAllowedAddressesByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the list of IP addresses and CIDR IP subnets that are always allowed access. This is a location specific function, any action will operate on the specified location.

```java
void setAllowedAddressesByLocation{
    String location
    String[] class_names
    String[][] values
}
```
void setAllowedAddressesByLocation(
    String location
    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
)

setBannedAddressesByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the list of IP addresses and CIDR IP subnets that aren't allowed access. This is a location specific function, any action will operate on the specified location.

void setBannedAddressesByLocation(
    String location
    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
)

setDebugByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

void setDebugByLocation(
    String location
    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
)
setEnabledByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether the service protection classes are enabled. This is a location specific function, any action will operate on the specified location.

```java
void setEnabledByLocation(
    String location,
    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.

```java
void setHTTPCheckRfc2396(
    String[] class_names,
    Boolean[] values
)
```

setHTTPCheckRfc2396ByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether requests with poorly-formed URLs (as specified in RFC 2396) should be rejected. This is a location specific function, any action will operate on the specified location.

```java
void setHTTPCheckRfc2396ByLocation(
    String location,
    String[] class_names,
    Boolean[] values
)
```

setHTTPRejectBinary( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether requests containing binary data (after decoding) should be rejected.

```java
void setHTTPRejectBinary(
    String[] class_names,
    Boolean[] values
)
```

setHTTPRejectBinaryByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set whether requests containing binary data (after decoding) should be rejected. This is a location specific function, any action will operate on the specified location.

```java
void setHTTPRejectBinaryByLocation(
    String location,
    String[] class_names,
    Boolean[] values
)
```
**setHTTPHeaderSendErrorPage( 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.

```java
void setHTTPHeaderSendErrorPage(
    String[] class_names
    Boolean[] values
)
```

**setHTTPHeaderSendErrorPageByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setHTTPHeaderSendErrorPageByLocation(
    String location
    String[] class_names
    Boolean[] values
)
```

**setLogInterval( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Set the interval between logging service protection messages (in seconds).

```java
void setLogInterval(
    String[] class_names
    Unsigned Integer[] values
)
```

**setLogIntervalByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Set the interval between logging service protection messages (in seconds). This is a location specific function, any action will operate on the specified location.

```java
void setLogIntervalByLocation(
    String location
    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.

```java
void setMax10Connections(
    String[] class_names
    Unsigned Integer[] values
)
```
setMax10ConnectionsByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum number of simultaneous connections allowed from the top 10 busiest IP addresses. This is a location specific function, any action will operate on the specified location.

```java
void setMax10ConnectionsByLocation(
    String location,
    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 (0 means unlimited).

```java
void setMax1Connections(
    String[] class_names,
    Unsigned Integer[] values)
```

setMax1ConnectionsByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum number of simultaneous connections allowed from an individual IP address (0 means unlimited). This is a location specific function, any action will operate on the specified location.

```java
void setMax1ConnectionsByLocation(
    String location,
    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).

```java
void setMaxConnectionRate(
    String[] class_names,
    Unsigned Integer[] values)
```

setMaxConnectionRateByLocation( location, 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). This is a location specific function, any action will operate on the specified location.

```java
void setMaxConnectionRateByLocation(
    String location,
    String[] class_names,
    Unsigned Integer[] values)
```
setMaxHTTPBodyLength( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of the HTTP request body data (in bytes, 0 means no limit).

```java
void setMaxHTTPBodyLength(
    String[] class_names
    Unsigned Integer[] values
)
```

setMaxHTTPBodyLengthByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of the HTTP request body data (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

```java
void setMaxHTTPBodyLengthByLocation(
    String location
    String[] class_names
    Unsigned Integer[] values
)
```

setMaxHTTPHeaderLength( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of a single HTTP request header (in bytes, 0 means no limit).

```java
void setMaxHTTPHeaderLength(
    String[] class_names
    Unsigned Integer[] values
)
```

setMaxHTTPHeaderLengthByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of a single HTTP request header (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

```java
void setMaxHTTPHeaderLengthByLocation(
    String location
    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).

```java
void setMaxHTTPRequestLength(
    String[] class_names
    Unsigned Integer[] values
)
```

setMaxHTTPRequestLengthByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the maximum size of all the HTTP request headers (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

```java
void setMaxHTTPRequestLengthByLocation(
    String location
    String[] class_names
    Unsigned Integer[] values
)
```
void setMaxHTTPRequestLengthByLocation(
    String location
    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
)

setMaxHTTPURLLengthByLocation( location, class_names, values ) throws
ObjectDoesNotExist, DeploymentError, InvalidInput
Set the maximum size of the request URL (in bytes, 0 means no limit). This is a location specific function, any action will operate on the specified location.

void setMaxHTTPURLLengthByLocation(
    String location
    String[] class_names
    Unsigned Integer[] values
)

setMaxConnections( 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 setMaxConnections(
    String[] class_names
    Unsigned Integer[] values
)

setMaxConnectionsByLocation( location, class_names, values ) throws
ObjectDoesNotExist, DeploymentError, InvalidInput
Set the number of simultaneous connections that are always allowed from each IP address (0 means unlimited). This is a location specific function, any action will operate on the specified location.

void setMaxConnectionsByLocation(
    String location
    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
)
**setPerProcessConnectionCount** *(class_names, values)* throws **ObjectDoesNotExist, DeploymentError, InvalidInput**

Set whether or not each process within a Traffic Manager counts connections independently, when limiting the maximum simultaneous connections allowed from one IP address.

```java
void setPerProcessConnectionCount(
    String[] class_names,
    Boolean[] values
)
```

**setPerProcessConnectionCountByLocation** *(location, class_names, values)* throws **ObjectDoesNotExist, DeploymentError, InvalidInput**

Set whether or not each process within a Traffic Manager counts connections independently, when limiting the maximum simultaneous connections allowed from one IP address. This is a location specific function, any action will operate on the specified location.

```java
void setPerProcessConnectionCountByLocation(
    String location,
    String[] class_names,
    Boolean[] 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. The valid range is 1-99999 seconds.

```java
void setRateTimer(
    String[] class_names,
    Unsigned Integer[] values
)
```

**setRateTimerByLocation** *(location, 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. The valid range is 1-99999 seconds. This is a location specific function, any action will operate on the specified location.

```java
void setRateTimerByLocation(
    String location,
    String[] class_names,
    Unsigned Integer[] values
)
```

**setRule** *(class_names, values)* throws **ObjectDoesNotExist, DeploymentError, InvalidInput**

Set the TrafficScript rule to be applied to all connections.

```java
void setRule(
    String[] class_names,
    String[] values
)
```
setRuleByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput

Set the TrafficScript rule to be applied to all connections. This is a location specific function, any action will operate on the specified location.

```java
void setRuleByLocation(
    String location
    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.

```java
void setTesting(
    String[] class_names
    Boolean[] values
)
```

setTestingByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setTestingByLocation(
    String location
    String[] class_names
    Boolean[] values
)
```

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.

Methods

addPersistence( class_names ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add new persistence classes.

```java
void addPersistence{
    String[] class_names
}
```
Function Reference

**copyPersistence( class_names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError**

Copy the named persistence classes.

```java
void copyPersistence(
    String[] class_names
    String[] new_names
)
```

**deletePersistence( class_names ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError**

Delete the named persistence classes.

```java
void deletePersistence(
    String[] class_names
)
```

**getCookie( class_names ) throws ObjectDoesNotExist**

Get the name of the cookie used to track session persistence.

```java
String[] getCookie(
    String[] class_names
)
```

**getCookieByLocation( location, class_names ) throws ObjectDoesNotExist**

Get the name of the cookie used to track session persistence. This is a location specific function, any action will operate on the specified location.

```java
String[] getCookieByLocation(
    String location
    String[] class_names
)
```

**getDelete( class_names ) throws ObjectDoesNotExist**

Get whether the session should be deleted if a failure occurs.

```java
Boolean[] getDelete(
    String[] class_names
)
```

**getDeleteByLocation( location, class_names ) throws ObjectDoesNotExist**

Get whether the session should be deleted if a failure occurs. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getDeleteByLocation(
    String location
    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.

```java
Catalog.Persistance.FailureMode[] getFailureMode(
    String[] class_names
)
```
getFailureModeByLocation( location, class_names ) throws ObjectDoesNotExist

Get the action that should be taken if the session data is invalid or the node specified cannot be contacted. This is a location specific function, any action will operate on the specified location.

Catalog.Persistence.FailureMode[] getFailureModeByLocation(
   String location
   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()

getSubnetPrefixLengthV4( class_names ) throws ObjectDoesNotExist

Get when using IP-based session persistence, ensure all requests from this IPv4 subnet, specified as a prefix length, are sent to the same node.

Integer[] getSubnetPrefixLengthV4(  
   String[] class_names
)

getSubnetPrefixLengthV4ByLocation( location, class_names ) throws ObjectDoesNotExist

Get when using IP-based session persistence, ensure all requests from this IPv4 subnet, specified as a prefix length, are sent to the same node. This is a location specific function, any action will operate on the specified location.

Integer[] getSubnetPrefixLengthV4ByLocation(  
   String location  
   String[] class_names
)

getSubnetPrefixLengthV6( class_names ) throws ObjectDoesNotExist

Get when using IP-based session persistence, ensure all requests from this IPv6 subnet, specified as a prefix length, are sent to the same node.

Integer[] getSubnetPrefixLengthV6(  
   String[] class_names
)
getSubnetPrefixLengthV6ByLocation( location, class_names ) throws ObjectDoesNotExist

Get when using IP-based session persistence, ensure all requests from this IPv6 subnet, specified as a prefix length, are sent to the same node. This is a location specific function, any action will operate on the specified location.

Integer[] getSubnetPrefixLengthV6ByLocation(
    String location
    String[] class_names
)

g getType( class_names ) throws ObjectDoesNotExist

Gets the session method type.

Catalog.Persistence.Type[] getType(
    String[] class_names
)

g getTypeByLocation( location, class_names ) throws ObjectDoesNotExist

Gets the session method type. This is a location specific function, any action will operate on the specified location.

Catalog.Persistence.Type[] getTypeByLocation(
    String location
    String[] class_names
)

g 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
)

g getUrlByLocation( location, class_names ) throws ObjectDoesNotExist

Get the URL to send to clients if the session persistence is configured to redirect users when a node dies. This is a location specific function, any action will operate on the specified location.

String[] getUrlByLocation(
    String location
    String[] class_names
)

g 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.

```java
void setCookie(
    String[] class_names
    String[] values
)
```

setCookieByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput
Set the name of the cookie used to track session persistence. This is a location specific function, any action will operate on the specified location.

```java
void setCookieByLocation(
    String location
    String[] class_names
    String[] values
)
```

setDelete( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput
Set whether the session should be deleted if a failure occurs.

```java
void setDelete(
    String[] class_names
    Boolean[] values
)
```

setDeleteByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput
Set whether the session should be deleted if a failure occurs. This is a location specific function, any action will operate on the specified location.

```java
void setDeleteByLocation(
    String location
    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.

```java
void setFailureMode(
    String[] class_names
    Catalog.Persistence.FailureMode[] values
)
```

setFailureModeByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setFailureModeByLocation(
    String location
    String[] class_names
    Catalog.Persistence.FailureMode[] values
)
```
setFailureModeByLocation

Function Reference

```java
void setFailureModeByLocation(
    String location
    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.

```java
void setNote(
    String[] class_names
    String[] values
)
```

`setSubnetPrefixLengthV4( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput`

Set when using IP-based session persistence, ensure all requests from this IPv4 subnet, specified as a prefix length, are sent to the same node.

```java
void setSubnetPrefixLengthV4(
    String[] class_names
    Integer[] values
)
```

`setSubnetPrefixLengthV4ByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput`

Set when using IP-based session persistence, ensure all requests from this IPv4 subnet, specified as a prefix length, are sent to the same node. This is a location specific function, any action will operate on the specified location.

```java
void setSubnetPrefixLengthV4ByLocation(
    String location
    String[] class_names
    Integer[] values
)
```

`setSubnetPrefixLengthV6( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput`

Set when using IP-based session persistence, ensure all requests from this IPv6 subnet, specified as a prefix length, are sent to the same node.

```java
void setSubnetPrefixLengthV6(
    String[] class_names
    Integer[] values
)
```

`setSubnetPrefixLengthV6ByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput`

Set when using IP-based session persistence, ensure all requests from this IPv6 subnet, specified as a prefix length, are sent to the same node. This is a location specific function, any action will operate on the specified location.

```java
void setSubnetPrefixLengthV6ByLocation(
```

308
**Catalog.Persistence Function Reference**

**String location**

**String[] class_names**

**Integer[] values**

)  

**setType( class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Sets the session method type.

```java
void setType(
    String[] class_names
    Catalog.Persistence.Type[] values
)
```

**setTypeByLocation( location, class_names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Sets the session method type. This is a location specific function, any action will operate on the specified location.

```java
void setTypeByLocation(
    String location
    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.

```java
void setUrl(
    String[] class_names
    String[] values
)
```

**setUrlByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
void setUrlByLocation(
    String location
    String[] class_names
    String[] values
)
```

**Enumerations**

**Catalog.Persistence.FailureMode**

```java
enum Catalog.Persistence.FailureMode {
    # Choose a new node to use
    newnode,

    # Redirect the user to a given URL
}
```
# Catalog.Persistence.Type

e num 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
}
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
)

getMaximumByLocation( location, class_names ) throws ObjectDoesNotExist, LicenseError

Get the maximum bandwidth, in kbits/second. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getMaximumByLocation(
    String location
    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
)

getSharingByLocation( location, class_names ) throws ObjectDoesNotExist, LicenseError

Get the bandwidth sharing mode. This is a location-specific function, any action will operate on the specified location.

Catalog.Bandwidth.Sharing[] getSharingByLocation(
    String location
    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
)

setMaximumByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the maximum bandwidth, in kbits/second. This is a location-specific function, any action will operate on the specified location.

void setMaximumByLocation(
    String location
    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

```java
void setSharing(
    String[] class_names
    Catalog.Bandwidth.Sharing[] values
)
```

setSharingByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the bandwidth sharing mode This is a location specific function, any action will operate on the specified location.

```java
void setSharingByLocation(
    String location
    String[] class_names
    Catalog.Bandwidth.Sharing[] values
)
```

Enumerations

Catalog.Bandwidth.Sharing

```java
enum Catalog.Bandwidth.Sharing {
    # Each connection can use the maximum rate connection,
    connection,

    # Bandwidth is shared per traffic manager machine,
    machine,

    # Bandwidth is shared across all traffic managers cluster
}
```

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.

Methods

addSLM( class_names ) throws InvalidObjectName, ObjectAlreadyExists, DeploymentError, LicenseError

Add new SLM classes.

```java
void addSLM(
```
Function Reference

```java
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
)

getResponseTimeByLocation( location, class_names ) throws ObjectDoesNotExist, LicenseError
Get the time limit for a response to conform (in milliseconds). This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getResponseTimeByLocation(
    String location
    String[] class_names
)

getSLMNames() throws LicenseError
Get the names of all the configured SLM classes.
String[] getSLMNames()

generateSeriousThreshold( 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
}

getSeriousThresholdByLocation( location, class_names ) throws ObjectDoesNotExist, LicenseError

Get the percentage of conforming responses below which a serious error will be emitted. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getSeriousThresholdByLocation{
    String location
    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
}

getWarningThresholdByLocation( location, class_names ) throws ObjectDoesNotExist, LicenseError

Get the percentage of conforming responses below which a warning message will be triggered. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getWarningThresholdByLocation{
    String location
    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
)

setResponseTimeByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the time limit for a response to conform (in milliseconds). This is a location specific function, any action will operate on the specified location.

void setResponseTimeByLocation(
    String location
    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
)

setSeriousThresholdByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the percentage of conforming responses below which a serious error will be emitted. This is a location specific function, any action will operate on the specified location.

void setSeriousThresholdByLocation(
    String location
    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
)

setWarningThresholdByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the percentage of conforming responses below which a warning message will be triggered. This is a location specific function, any action will operate on the specified location.

void setWarningThresholdByLocation(
    String location
    String[] class_names
    Unsigned Integer[] values
)
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.

Methods

**addRate( class_names ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError**

Add new rate classes.

```java
void addRate(
    String[] class_names
)
```

**copyRate( class_names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError**

Copy the named rate classes.

```java
void copyRate(
    String[] class_names
    String[] new_names
)
```

**deleteRate( class_names ) throws ObjectInUse, ObjectDoesNotExist, DeploymentError**

Delete the named rate classes.

```java
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.

```java
Unsigned Integer[] getMaxRatePerMinute(
    String[] class_names
)
```

**getMaxRatePerMinuteByLocation( location, class_names ) throws ObjectDoesNotExist**

Get the maximum rate at which requests are allowed to be processed, in requests per minute. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getMaxRatePerMinuteByLocation(
    String location
    String[] class_names
)
```
**getMaxRatePerSecond( class_names ) throws ObjectDoesNotExist**

Get the maximum rate at which requests are allowed to be processed, in requests per second.

```java
Unsigned Integer[] getMaxRatePerSecond(
    String[] class_names
)
```

**getMaxRatePerSecondByLocation( location, class_names ) throws ObjectDoesNotExist**

Get the maximum rate at which requests are allowed to be processed, in requests per second. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getMaxRatePerSecondByLocation(
    String location
    String[] class_names
)
```

**getNote( class_names ) throws ObjectDoesNotExist**

Get the note for each of the named Rate classes.

```java
String[] getNote(
    String[] class_names
)
```

**getRateNames()**

Get the names of all the configured rate classes.

```java
String[] getRateNames()
```

**renameRate( class_names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation**

Rename the named rate classes.

```java
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.

```java
void setMaxRatePerMinute(
    String[] class_names
    Unsigned Integer[] values
)
```

**setMaxRatePerMinuteByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the maximum rate at which requests are allowed to be processed, in requests per minute. This is a location specific function, any action will operate on the specified location.

```java
void setMaxRatePerMinuteByLocation(
```
setMaxRatePerSecond( class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum rate at which requests are allowed to be processed, in requests per second.

```java
void setMaxRatePerSecond(
    String[] class_names,
    Unsigned Integer[] values
)
```

setMaxRatePerSecondByLocation( location, class_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the maximum rate at which requests are allowed to be processed, in requests per second. This is a location specific function, any action will operate on the specified location.

```java
void setMaxRatePerSecondByLocation(
    String location,
    String[] class_names,
    Unsigned Integer[] values
)
```

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

Set the note for each of the named Rate classes.

```java
void setNote(
    String[] class_names,
    String[] values
)
```

---

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.

**Methods**

**addProperties( class_names, properties ) throws LicenseError, ObjectDoesNotExist, InvalidInput**

Adds initialisation properties for each of the specified extensions.

```java
void addProperties(
    String[] class_names,
    Catalog.JavaExtension.Property[][] properties
)
deleteJavaExtensionFile(names) throws LicenseError, ObjectDoesNotExist, ObjectInUse

Delete the named Java Extension files.

```java
void deleteJavaExtensionFile(
    String[] names
)
```

downloadJavaExtensionFile(name) throws LicenseError, ObjectDoesNotExist

Download the named Java Extension File.

```java
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.

```java
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.

```java
String[] getExtensionClassNames()
```

getExtensionErrors(class_names) throws LicenseError, ObjectDoesNotExist

Gets the errors for each of the specified extensions.

```java
String[][] getExtensionErrors(
    String[] class_names
)
```

getExtensionInfo(class_names) throws LicenseError, ObjectDoesNotExist

Gets information on each of the specified extensions.

```java
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.

```java
String[] getJavaExtensionFileNames()
```
**getProperties( class_names ) throws LicenseError, ObjectDoesNotExist**

Gets the initialisation properties for each of the specified extensions.

```java
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.

```java
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.

```java
void uploadJavaExtensionFile(
    String name
    Binary Data content
)
```

---

**Structures**

**Catalog.JavaExtension.Info**

This structure contains basic information about a Java Extension in the catalog.

```java
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.

```java
struct Catalog.JavaExtension.Property {
    # The name of this property
    String name;

    # The value of this property
    String value;
}
```
Catalog.Authenticators

URI: http://soap.zeus.com/zxtm/1.0/Catalog/Authenticators/

The Catalog.Authenticator interface allows management of authenticator information, which are used by TrafficScript to communicate with an authentication service.

Methods

addAuthenticator( class_names ) throws InvalidObjectName, ObjectAlreadyExists, DeploymentError, LicenseError
Add new Authenticator classes.

```java
void addAuthenticator(
    String[] class_names
)
```

addLDAPSearchAttr( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Add the attributes to return from the search.

```java
void addLDAPSearchAttr(
    String[] names
    String[][] values
)
```

addLDAPSearchAttrByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Add the attributes to return from the search. This is a location specific function, any action will operate on the specified location.

```java
void addLDAPSearchAttrByLocation(
    String location
    String[] names
    String[][] values
)
```

copyAuthenticator( class_names, new_names ) throws ObjectAlreadyExists, InvalidObjectName, ObjectDoesNotExist, DeploymentError, LicenseError
Copy the named Authenticator classes.

```java
void copyAuthenticator(
    String[] class_names
    String[] new_names
)
```

deleteAuthenticator( class_names ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, LicenseError
Delete the named Authenticator classes.

```java
void deleteAuthenticator(
```
String[] class_names

getAuthenticatorNames() throws LicenseError
Get the names of all the configured Authenticator classes.
String[] getAuthenticatorNames()

getHost( names ) throws ObjectDoesNotExist, LicenseError
Get the hostname of the remote authenticator.
String[] getHost(
  String[] names
)

getHostByLocation( location, names ) throws ObjectDoesNotExist, LicenseError
Get the hostname of the remote authenticator. This is a location specific function, any action will operate on
the specified location.
String[] getHostByLocation(
  String location
  String[] names
)

getLDAPBindDN( names ) throws ObjectDoesNotExist, LicenseError
Get the user used to connect to the LDAP server for each of the named Authenticators
String[] getLDAPBindDN(
  String[] names
)

getLDAPBindDNByLocation( location, names ) throws ObjectDoesNotExist, LicenseError
Get the user used to connect to the LDAP server for each of the named Authenticators. This is a location
specific function, any action will operate on the specified location.
String[] getLDAPBindDNByLocation(
  String location
  String[] names
)

getLDAPFilter( names ) throws ObjectDoesNotExist, LicenseError
Get the filter used to identify user records. Any occurrences of '%u' in the filter will be replaced by the name
of the user being authenticated.
String[] getLDAPFilter(
  String[] names
)

getLDAPFilterBaseDN( names ) throws ObjectDoesNotExist, LicenseError
Get the DN that we will search for user records under.
String[] getLDAPFilterBaseDN(
  String[] names
)
getLDAPFilterBaseDNByLocation( location, names ) throws ObjectDoesNotExist, LicenseError

Get the DN that we will search for user records under. This is a location specific function, any action will operate on the specified location.

```java
String[] getLDAPFilterBaseDNByLocation(
    String location
    String[] names
)
```

getLDAPFilterByLocation( location, names ) throws ObjectDoesNotExist, LicenseError

Get the filter used to identify user records. Any occurrences of '%u' in the filter will be replaced by the name of the user being authenticated. This is a location specific function, any action will operate on the specified location.

```java
String[] getLDAPFilterByLocation(
    String location
    String[] names
)
```

getLDAPSSLCertificate( names ) throws ObjectDoesNotExist, LicenseError

Get the SSL certificate in the CA catalog used to authenticate the remote LDAP server.

```java
String[] getLDAPSSLCertificate(
    String[] names
)
```

getLDAPSSLCertificateByLocation( location, names ) throws ObjectDoesNotExist, LicenseError

Get the SSL certificate in the CA catalog used to authenticate the remote LDAP server. This is a location specific function, any action will operate on the specified location.

```java
String[] getLDAPSSLCertificateByLocation(
    String location
    String[] names
)
```

getLDAPSSLEnabled( names ) throws ObjectDoesNotExist, LicenseError

Get if SSL should be used to connect to the LDAP server.

```java
Boolean[] getLDAPSSLEnabled(
    String[] names
)
```

getLDAPSSLEnabledByLocation( location, names ) throws ObjectDoesNotExist, LicenseError

Get if SSL should be used to connect to the LDAP server. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getLDAPSSLEnabledByLocation(
```
getLDAPSSLType( names ) throws ObjectDoesNotExist, LicenseError

Get how a SSL connection should be established.

```java
Catalog.Authenticators.LDAPSSLType[] getLDAPSSLType(
    String[] names
)
```

getLDAPSSLTypeByLocation( location, names ) throws ObjectDoesNotExist, LicenseError

Get how a SSL connection should be established. This is a location specific function, any action will operate on the specified location.

```java
Catalog.Authenticators.LDAPSSLType[] getLDAPSSLTypeByLocation(
    String location,
    String[] names
)
```

getLDAPSearchAttr( names ) throws ObjectDoesNotExist, LicenseError

Get the attributes to return from the search.

```java
String[][] getLDAPSearchAttr(
    String[] names
)
```

getLDAPSearchAttrByLocation( location, names ) throws ObjectDoesNotExist, LicenseError

Get the attributes to return from the search. This is a location specific function, any action will operate on the specified location.

```java
String[][] getLDAPSearchAttrByLocation(
    String location,
    String[] names
)
```

getNote( names ) throws ObjectDoesNotExist, LicenseError

Get the note for each of the named Authenticators

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

getPort( names ) throws ObjectDoesNotExist, LicenseError

Get the port of the remote authenticator.

```java
Unsigned Integer[] getPort(
    String[] names
)
```
**Function Reference Catalog: Authenticators**

**getPortByLocation( location, names ) throws ObjectDoesNotExist, LicenseError**

Get the port of the remote authenticator. This is a location specific function, any action will operate on the specified location.

```
Unsigned Integer[] getPortByLocation(
    String location
    String[] names
)
```

**removeLDAPSearchAttr( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Remove the attributes to return from the search.

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

**removeLDAPSearchAttrByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Remove the attributes to return from the search. This is a location specific function, any action will operate on the specified location.

```
void removeLDAPSearchAttrByLocation(
    String location
    String[] names
    String[][] values
)
```

**renameAuthenticator( class_names, new_names ) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, InvalidOperation, DeploymentError, LicenseError**

Rename the named Authenticator classes.

```
void renameAuthenticator(
    String[] class_names
    String[] new_names
)
```

**setHost( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Set the hostname of the remote authenticator.

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

**setHostByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Set the hostname of the remote authenticator. This is a location specific function, any action will operate on the specified location.

```
void setHostByLocation(
    String location
    String[] names
    String[] values
)
```
String location
String[] names
String[] values
)

setLDAPBindDN(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the user used to connect to the LDAP server for each of the named Authenticators.

void setLDAPBindDN(
    String[] names
    String[] values
)

setLDAPBindDNByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the user used to connect to the LDAP server for each of the named Authenticators. This is a location specific function. Any action will operate on the specified location.

void setLDAPBindDNByLocation(
    String location
    String[] names
    String[] values
)

setLDAPBindPassword(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the password of the bind user.

void setLDAPBindPassword(
    String[] names
    String[] values
)

setLDAPBindPasswordByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the password of the bind user. This is a location specific function. Any action will operate on the specified location.

void setLDAPBindPasswordByLocation(
    String location
    String[] names
    String[] values
)

setLDAPFilter(names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the filter used to identify user records. Any occurrences of '%u' in the filter will be replaced by the name of the user being authenticated.

void setLDAPFilter(
    String[] names
    String[] values
)
setLDAPFilterBaseDN( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the DN that we will search for user records under.

```java
void setLDAPFilterBaseDN(
    String[] names
    String[] values
)
```

setLDAPFilterBaseDNByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the DN that we will search for user records under. This is a location specific function, any action will operate on the specified location.

```java
void setLDAPFilterBaseDNByLocation(
    String location
    String[] names
    String[] values
)
```

setLDAPFilterByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the filter used to identify user records. Any occurrences of '%u' in the filter will be replaced by the name of the user being authenticated. This is a location specific function, any action will operate on the specified location.

```java
void setLDAPFilterByLocation(
    String location
    String[] names
    String[] values
)
```

setLDAPSSLCertificate( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the SSL certificate in the CA catalog used to authenticate the remote LDAP server.

```java
void setLDAPSSLCertificate(
    String[] names
    String[] values
)
```

setLDAPSSLCertificateByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the SSL certificate in the CA catalog used to authenticate the remote LDAP server. This is a location specific function, any action will operate on the specified location.

```java
void setLDAPSSLCertificateByLocation(
    String location
    String[] names
    String[] values
)
```
setLDAPSSLEnabled( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set if SSL should be used to connect to the LDAP server.

```java
void setLDAPSSLEnabled(
    String[] names
    Boolean[] values
)
```

setLDAPSSLEnabledByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set if SSL should be used to connect to the LDAP server. This is a location specific function, any action will operate on the specified location.

```java
void setLDAPSSLEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```

setLDAPSSLType( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set how a SSL connection should be established.

```java
void setLDAPSSLType(
    String[] names
    Catalog.Authenticators.LDAPSSLType[] values
)
```

setLDAPSSLTypeByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set how a SSL connection should be established. This is a location specific function, any action will operate on the specified location.

```java
void setLDAPSSLTypeByLocation(
    String location
    String[] names
    Catalog.Authenticators.LDAPSSLType[] values
)
```

setLDAPSearchAttr( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the attributes to return from the search.

```java
void setLDAPSearchAttr(
    String[] names
    String[][] values
)
```

setLDAPSearchAttrByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the attributes to return from the search. This is a location specific function, any action will operate on the specified location.
void setLDAPSearchAttrByLocation(
    String location
    String[] names
    String[][] values
)

setNote( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the note for each of the named Authenticators
void setNote{
    String[] names
    String[] values
}

setPort( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the port of the remote authenticator.
void setPort{
    String[] names
    Unsigned Integer[] values
}

setPortByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the port of the remote authenticator. This is a location specific function, any action will operate on the specified location.
void setPortByLocation{
    String location
    String[] names
    Unsigned Integer[] values
}

Enumerations

Catalog.Authenticators.LDAPSSLType
eenum Catalog.Authenticators.LDAPSSLType {
    # LDAPS
    ldaps,
    # Start TLS
    starttls
}

Catalog.DNSServer.ZoneFiles
URI: http://soap.zeus.com/zxtm/1.0/Catalog/DNSServer/ZoneFiles/
The Catalog.DNSServer.ZoneFiles interface allows management of the DNS zone files stored in the conf/dnsserver/zonefiles directory.

**Methods**

**deleteFile( names ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError**

Delete the named DNS zone files from the conf/dnsserver/zonefiles directory.

```java
void deleteFile(
  String[] names
)
```

**downloadFile( name ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError**

Download the named configuration file from the conf/dnsserver/zonefiles directory.

```java
Binary Data downloadFile(
  String name
)
```

**getFileNames()**

Get the names of all the DNS zone files stored in the conf/dnsserver/zonefiles directory.

```java
String[] getFileNames()
```

**uploadFile( name, content ) throws InvalidObjectName, DeploymentError**

Uploads a new DNS zone file into the conf/dnsserver/zonefiles, overwriting the zone file if it already exists.

```java
void uploadFile(
  String name
  Binary Data content
)
```

---

**Catalog.DNSServer.Zones**

URI: http://soap.zeus.com/zxtm/1.0/Catalog/DNSServer/Zones/

The Catalog.DNSServer.Zones interface allows management of DNS zones. Using this interface, you can create, delete and rename DNS zones, and manage their configuration.

**Methods**

**addZone( zone_names, zone_parameters ) throws InvalidObjectName, InvalidInput, ObjectAlreadyExists, DeploymentError**

Add new DNS zone.

```java
void addZone(
  String[] zone_names
  Catalog.DNSServer.Zones.DNSZoneParameter[] zone_parameters
)
Function Reference Catalog.DNSServer.Zones

}  
copyZone( zone_names, new_names ) throws ObjectAlreadyExists, 
InvalidObjectName, ObjectDoesNotExist, DeploymentError

Copy the named DNS zones.

void copyZone{
    String[] zone_names
    String[] new_names
}

deleteZone( zone_names ) throws ObjectDoesNotExist, ObjectInUse, 
DeploymentError

Delete the named DNS zone.

void deleteZone{
    String[] zone_names
}

getOrigin( zone_names ) throws ObjectDoesNotExist

Get Zone origin

String[] getOrigin{
    String[] zone_names
}

getZoneNames()

Get the names of all the configured DNS zones

String[] getZoneNames()

getZonefile( zone_names ) throws ObjectDoesNotExist

Get Zone file

String[] getZonefile{
    String[] zone_names
}

renameZone( zone_names, new_names ) throws ObjectAlreadyExists, 
ObjectDoesNotExist, InvalidObjectName, InvalidOperation, DeploymentError

Rename the named DNS zones.

void renameZone{
    String[] zone_names
    String[] new_names
}

setOrigin( zone_names, values ) throws ObjectDoesNotExist, InvalidInput, 
DeploymentError

Set Zone origin

void setOrigin{
    String[] zone_names
}
GlobalSettings

URI: http://soap.zeus.com/zxtm/1.0/GlobalSettings/

The Global Settings interface allows management of the traffic manager settings.

Methods

addApplianceReturnPathRoutes( value ) throws InvalidInput, DeploymentError

Add a set of return path routes (MAC/IP mappings) to the configuration.

void addApplianceReturnPathRoutes(
   GlobalSettings.ReturnPathRoute[] value
)

addApplianceReturnPathRoutesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Add a set of return path routes (MAC/IP mappings) to the configuration. This is a location specific function, any action will operate on the specified location.

void addApplianceReturnPathRoutesByLocation(
   String location
   GlobalSettings.ReturnPathRoute[] value
)
addFlipperFrontendCheckAddresses(values) throws InvalidInput, DeploymentError

Add new IP addresses to the list that should be used to check front-end connectivity

```java
void addFlipperFrontendCheckAddresses(
    String[] values
)
```

addFlipperFrontendCheckAddressesByLocation(location, values) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Add new IP addresses to the list that should be used to check front-end connectivity. This is a location specific function, any action will operate on the specified location.

```java
void addFlipperFrontendCheckAddressesByLocation(
    String location
    String[] values
)
```

addLicenseServers(values) throws InvalidInput, DeploymentError

Add a list of license servers for FLA licensing.

```java
void addLicenseServers(
    String[] values
)
```

addLicenseServersByLocation(location, values) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Add a list of license servers for FLA licensing. This is a location specific function, any action will operate on the specified location.

```java
void addLicenseServersByLocation(
    String location
    String[] values
)
```

getASPSessionCacheSize()

Get the maximum number of entries in the ASP session cache.

```java
Unsigned Integer getASPSessionCacheSize()
```

getASPSessionCacheSizeByLocation(location) throws ObjectDoesNotExist

Get the maximum number of entries in the ASP session cache. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getASPSessionCacheSizeByLocation(
    String location
)
```

getAcceptingDelay()

Get how often each traffic manager child process checks whether it should be accepting new connections.

```java
Unsigned Integer getAcceptingDelay()
```
getAcceptingDelayByLocation( location ) throws ObjectDoesNotExist

Get how often each traffic manager child process checks whether it should be accepting new connections. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getAcceptingDelayByLocation(
    String location
)

getAdminAllowRehandshake()

Get whether SSL / TLS re-handshakes are supported.

GlobalSettings.AdminAllowRehandshake getAdminAllowRehandshake()

getAdminDiffieHellmanKeyLength()

Get the number of bits to use for Diffie-Hellman keys

GlobalSettings.AdminDiffieHellmanKeyLength getAdminDiffieHellmanKeyLength()

getAdminHonorFallbackSCSV()

Get whether admin server, internal control port and config daemon honor the Fallback SCSV

Boolean getAdminHonorFallbackSCSV()

getAdminInsertExtraFragment()

Get whether admin server SSL3 and TLS1 use one byte fragments

Boolean getAdminInsertExtraFragment()

getAdminMinRehandshakeInterval()

Get the minimum time interval (in milliseconds) between handshakes on a single SSL3/TLS connection.

Unsigned Integer getAdminMinRehandshakeInterval()

getAdminSSLCiphers()

Get the list of configured SSL ciphers for admin server and internal connections (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s).

String getAdminSSLCiphers()

getAdminSSLEllipticCurves()

Get the elliptic curve preference list for SSL connections to the admin server and within the traffic manager cluster.

String getAdminSSLEllipticCurves()

getAdminSSLMaxHandshakeMessageSize()

Get the maximum acceptable size (in bytes) a SSL handshake message is permitted to be for admin and internal connections.

Unsigned Integer getAdminSSLMaxHandshakeMessageSize()
getAdminSSLPreventTimingSideChannels()
Get whether SSL3 and TLS used by the admin server and internal connections will take performance
degrading steps to prevent exposing timing side-channels.
Boolean getAdminSSLPreventTimingSideChannels()

getAdminSSLSignatureAlgorithms()
Get the SSL signature algorithms preference list for SSL connections to the admin server and within the
zttm cluster.
String getAdminSSLSignatureAlgorithms()

getAdminSSLSupportTLS11()
Get whether TLSv1.1 support is enabled for admin server and internal connections.
Boolean getAdminSSLSupportTLS11()

getAdminSSLSupportTLS12()
Get whether TLSv1.2 support is enabled for admin server and internal connections.
Boolean getAdminSSLSupportTLS12()

getAdminSupportSSL2()
This method is now deprecated.
Boolean getAdminSupportSSL2()

getAdminSupportSSL3()
Get whether SSLv3 support is enabled for admin server and internal connections.
Boolean getAdminSupportSSL3()

getAdminSupportTLS1()
Get whether TLSv1 support is enabled for admin server and internal connections.
Boolean getAdminSupportTLS1()

getAfmEnabled()
Get whether the Application Firewall is enabled
Boolean getAfmEnabled()

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()
**getAlertEmailIntervalByLocation( location )** throws **ObjectDoesNotExist**

Get the length of time between alert emails, in seconds. Several alert messages will be stored up and sent in one email. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getAlertEmailIntervalByLocation(
    String location
)
```

**getAlertEmailMaxAttempts()**

Get the number of times to attempt sending an email before giving up.

```java
Unsigned Integer getAlertEmailMaxAttempts()
```

**getAlertEmailMaxAttemptsByLocation( location )** throws **ObjectDoesNotExist**

Get the number of times to attempt sending an email before giving up. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getAlertEmailMaxAttemptsByLocation(
    String location
)
```

**getAllowConsecutiveChars()**

Get whether the same character can appear consecutively in passwords.

```java
Boolean getAllowConsecutiveChars()
```

**getApplianceReturnPathRoutes()**

Get the set of return path routes (MAC/IP mappings) in the configuration.

```java
GlobalSettings.ReturnPathRoute[] getApplianceReturnPathRoutes()
```

**getApplianceReturnPathRoutesByLocation( location )** throws **ObjectDoesNotExist**

Get the set of return path routes (MAC/IP mappings) in the configuration. This is a location specific function, any action will operate on the specified location.

```java
GlobalSettings.ReturnPathRoute[] getApplianceReturnPathRoutesByLocation(
    String location
)
```

**getApplianceReturnPathRoutingEnabled()**

Get whether return path routing is enabled

```java
Boolean getApplianceReturnPathRoutingEnabled()
```

**getApplianceReturnPathRoutingEnabledByLocation( location )** throws **ObjectDoesNotExist**

Get whether return path routing is enabled. This is a location specific function, any action will operate on the specified location.

```java
Boolean getApplianceReturnPathRoutingEnabledByLocation(
    String location
)
```
**getAptimizerMaxDependentFetchSize()**
Get the maximum size of a dependent resource that can be sent to Web Accelerator. Set to 0 to disable limit.

```java
String getAptimizerMaxDependentFetchSize()
```

**getAptimizerMaxOriginalContentBufferSize()**
Get the maximum size of original content buffer for content sent to Web Accelerator.

```java
String getAptimizerMaxOriginalContentBufferSize()
```

**getAptimizerWatchdogInterval()**
Get How long (in seconds) the Web Accelerator watchdog mechanism should keep count of crashes for.

```java
Unsigned Integer getAptimizerWatchdogInterval()
```

**getAptimizerWatchdogLimit()**
Get the maximum number of times the Web Accelerator sub-process will be restarted.

```java
Unsigned Integer getAptimizerWatchdogLimit()
```

**getAuditlogViaEventd()**
Get whether the auditlog is to be mirrored to EventD.

```java
Boolean getAuditlogViaEventd()
```

**getAuditlogViaEventdByLocation(location) throws ObjectDoesNotExist**
Get whether the auditlog is to be mirrored to EventD. This is a location specific function, any action will operate on the specified location.

```java
Boolean getAuditlogViaEventdByLocation(String location)
```

**getAuditlogViaSyslog()**
Get whether the auditlog is to be mirrored to the syslog

```java
Boolean getAuditlogViaSyslog()
```

**getAuditlogViaSyslogByLocation(location) throws ObjectDoesNotExist**
Get whether the auditlog is to be mirrored to the syslog This is a location specific function, any action will operate on the specified location.

```java
Boolean getAuditlogViaSyslogByLocation(String location)
```

**getAutoscalerVerbose()**
Get detailed logging of autoscaler status and actions

```java
Boolean getAutoscalerVerbose()
```
GlobalSettings Function Reference

getAutoscalerVerboseByLocation( location ) throws ObjectDoesNotExist
Get detailed logging of autoscaler status and actions. This is a location specific function, any action will operate on the specified location.

Boolean getAutoscalerVerboseByLocation(
    String location
)

getBackendKeepaliveTimeout()
getBackendKeepaliveTimeout is deprecated, please use getIdleConnectionTimeout instead.
Unsigned Integer getBackendKeepaliveTimeout()

getBackendKeepaliveTimeoutByLocation( location ) throws ObjectDoesNotExist
getBackendKeepaliveTimeout is deprecated, please use getIdleConnectionTimeout instead. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getBackendKeepaliveTimeoutByLocation(
    String location
)

getBandwidthSharing()
This method is now obsolete and is replaced by Catalog.Bandwidth.getSharing.

Boolean getBandwidthSharing()

getBannerAccept()
Get whether or not users must explicitly agree to the displayed login_banner text before logging in to the Admin Server.

Boolean getBannerAccept()

getBgpAsNumber()
Get the number of the BGP AS in which the traffic manager will operate.

Unsigned Integer getBgpAsNumber()

getBgpAsNumberByLocation( location ) throws ObjectDoesNotExist
Get the number of the BGP AS in which the traffic manager will operate. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getBgpAsNumberByLocation(
    String location
)

getBgpEnabled()
Get whether BGP Route Health Injection is enabled.

Boolean getBgpEnabled()
Function Reference

GlobalSettings

getBgpEnabledByLocation( location ) throws ObjectDoesNotExist

Get whether BGP Route Health Injection is enabled. This is a location specific function, any action will operate on the specified location.

```java
Boolean getBgpEnabledByLocation(String location)
```

getChunkSize()

Get the default chunk size for reading and writing data, in bytes.

```java
Unsigned Integer getChunkSize()
```

getChunkSizeByLocation( location ) throws ObjectDoesNotExist

Get the default chunk size for reading and writing data, in bytes. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getChunkSizeByLocation(String location)
```

ggetClientFirstOpt()

Get whether client-first network socket optimisations should be used.

```java
Boolean getClientFirstOpt()
```

ggetClientFirstOptByLocation( location ) throws ObjectDoesNotExist

Get whether client-first network socket optimisations should be used. This is a location specific function, any action will operate on the specified location.

```java
Boolean getClientFirstOptByLocation(String location)
```

ggetControlAllowHosts()

Get the hosts that are allowed to contact the internal administration port on each traffic manager.

```java
String getControlAllowHosts()
```

ggetControlAllowHostsByLocation( location ) throws ObjectDoesNotExist

Get the hosts that are allowed to contact the internal administration port on each traffic manager. This is a location specific function, any action will operate on the specified location.

```java
String getControlAllowHostsByLocation(String location)
```

ggetControlCanUpdateDefault()

Get the value of the control!canupdate key for new cluster members.

```java
Boolean getControlCanUpdateDefault()
```
getDNSCacheExpiryTime()

This method is now deprecated and is replaced by getDNSCacheMaxTTL/getDNSCacheMinTTL.

Unsigned Integer getDNSCacheExpiryTime()

getDNSCacheExpiryTimeByLocation( location ) throws ObjectDoesNotExist

This method is now deprecated and is replaced by getDNSCacheMaxTTL/getDNSCacheMinTTL. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDNSCacheExpiryTimeByLocation(
  String location
)

getDNSCacheMaxTTL()

Get the maximum time entries are stored in the DNS cache for, in seconds.

Unsigned Integer getDNSCacheMaxTTL()

getDNSCacheMaxTTLByLocation( location ) throws ObjectDoesNotExist

Get the maximum time entries are stored in the DNS cache for, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDNSCacheMaxTTLByLocation(
  String location
)

getDNSCacheMinTTL()

Get the minimum time entries are stored in the DNS cache for, in seconds.

Unsigned Integer getDNSCacheMinTTL()

getDNSCacheMinTTLByLocation( location ) throws ObjectDoesNotExist

Get the minimum time entries are stored in the DNS cache for, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDNSCacheMinTTLByLocation(
  String location
)

getDNSCacheNegativeExpiryTime()

Get the time failed lookups are stored in the DNS cache for, in seconds.

Unsigned Integer getDNSCacheNegativeExpiryTime()

getDNSCacheNegativeExpiryTimeByLocation( location ) throws ObjectDoesNotExist

Get the time failed lookups are stored in the DNS cache for, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDNSCacheNegativeExpiryTimeByLocation(
  String location
)
**getDNSCacheSize()**
Get the maximum number of entries in the DNS cache.

Unsigned Integer getDNSCacheSize()

**getDNSCacheSizeByLocation( location ) throws ObjectDoesNotExist**
Get the maximum number of entries in the DNS cache. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDNSCacheSizeByLocation(
    String location
)

**getDNSTimeout()**
Get the timeout for receiving a response from a DNS Server, in seconds.

Unsigned Integer getDNSTimeout()

**getDNSTimeoutByLocation( location ) throws ObjectDoesNotExist**
Get the timeout for receiving a response from a DNS Server, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDNSTimeoutByLocation(
    String location
)

**getDataPlaneAccelerationCores()**
Get the number of CPU cores assigned to assist with data plane acceleration. These cores are dedicated to reading and writing packets to the network interface cards and distributing packets between the traffic manager processes.

GlobalSettings.DataPlaneAccelerationCores getDataPlaneAccelerationCores()

**getDataPlaneAccelerationCoresByLocation( location ) throws ObjectDoesNotExist**
Get the number of CPU cores assigned to assist with data plane acceleration. These cores are dedicated to reading and writing packets to the network interface cards and distributing packets between the traffic manager processes. This is a location specific function, any action will operate on the specified location.

GlobalSettings.DataPlaneAccelerationCores getDataPlaneAccelerationCoresByLocation(
    String location
)

**getDataPlaneAccelerationMode()**
Get whether Data Plane Acceleration Mode is enabled.

Boolean getDataPlaneAccelerationMode()

**getDataPlaneAccelerationModeByLocation( location ) throws ObjectDoesNotExist**
Get whether Data Plane Acceleration Mode is enabled. This is a location specific function, any action will operate on the specified location.

Boolean getDataPlaneAccelerationModeByLocation(}
getDataPlaneAccelerationTCPDelayAck()
Get the time, in milliseconds, to delay sending a TCP ACK response, providing an opportunity for additional data to be incorporated into the response and potentially improving network performance. The setting affects TCP connections handled by layer 7 services running in Data Plane Acceleration mode.

Unsigned Integer getDataPlaneAccelerationTCPDelayAck()

dataGetPlaneAccelerationTCPDelayAckByLocation( location ) throws ObjectDoesNotExist
Get the time, in milliseconds, to delay sending a TCP ACK response, providing an opportunity for additional data to be incorporated into the response and potentially improving network performance. The setting affects TCP connections handled by layer 7 services running in Data Plane Acceleration mode. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDataPlaneAccelerationTCPDelayAckByLocation(String location)

dataGetPlaneAccelerationTCPWinScale()
Get the TCP window scale option, which configures the size of the receive window for TCP connections handled by layer 7 services when running in Data Plane Acceleration mode.

Unsigned Integer getDataPlaneAccelerationTCPWinScale()

dataGetPlaneAccelerationTCPWinScaleByLocation( location ) throws ObjectDoesNotExist
Get the TCP window scale option, which configures the size of the receive window for TCP connections handled by layer 7 services when running in Data Plane Acceleration mode. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getDataPlaneAccelerationTCPWinScaleByLocation(String location)

dataGetDeadTime()
This method is now obsolete and is replaced by Pool.getNodeFailTime.

Unsigned Integer getDataDeadTime()

dataGetEC2AccessKeyID()
Get the Access Key ID used for interacting with the EC2 API.

String getDataEC2AccessKeyID()

dataGetEC2AccessKeyIDByLocation( location ) throws ObjectDoesNotExist
Get the Access Key ID used for interacting with the EC2 API. This is a location specific function, any action will operate on the specified location.

String getDataEC2AccessKeyIDByLocation(String location)
getEC2AwstoolTimeout()
Get the timeout for awstool requests to the AWS Query Server
Unsigned Integer getEC2AwstoolTimeout()

getEC2AwstoolTimeoutByLocation( location ) throws ObjectDoesNotExist
Get the timeout for awstool requests to the AWS Query Server. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getEC2AwstoolTimeoutByLocation( String location )

getEC2Endpoint()
Get URL for the Amazon EC2 AWS endpoint.
String getEC2Endpoint()

getEC2EndpointByLocation( location ) throws ObjectDoesNotExist
Get URL for the Amazon EC2 AWS endpoint. This is a location specific function, any action will operate on the specified location.
String getEC2EndpointByLocation( String location )

getEC2MetadataServer()
Get URL for the EC2 metadata server.
String getEC2MetadataServer()

getEC2MetadataServerByLocation( location ) throws ObjectDoesNotExist
Get URL for the EC2 metadata server. This is a location specific function, any action will operate on the specified location.
String getEC2MetadataServerByLocation( String location )

getEC2VerifyEndpointCert()
Get Whether to verify Amazon EC2 endpoint’s certificate using CAs present in SSL Certificate Authorities Catalog.
Boolean getEC2VerifyEndpointCert()

getEC2VerifyEndpointCertByLocation( location ) throws ObjectDoesNotExist
Get Whether to verify Amazon EC2 endpoint’s certificate using CAs present in SSL Certificate Authorities Catalog. This is a location specific function, any action will operate on the specified location.
GlobalSettings

```
Boolean getEC2VerifyEndpointCertByLocation(
    String location
)

getErrorLevel()
This method is now deprecated.
GlobalSettings.ErrorLevel getErrorLevel()

getErrorLevelByLocation( location ) throws ObjectDoesNotExist
This method is now deprecated. This is a location specific function, any action will operate on the specified location.
GlobalSettings.ErrorLevel getErrorLevelByLocation(
    String location
)

getErrorLogFile()
Get the filename that errors are logged to.
String getErrorLogFile()

getErrorLogFileByLocation( location ) throws ObjectDoesNotExist
Get the filename that errors are logged to. This is a location specific function, any action will operate on the specified location.
String getErrorLogFileByLocation(
    String location
)

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()

getFTPDataBindLowByLocation( location ) throws ObjectDoesNotExist
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. This is a location specific function, any action will operate on the specified location.
Boolean getFTPDataBindLowByLocation(
    String location
)

getFipsEnabled()
Get whether FIPS Mode is enabled.
Boolean getFipsEnabled()
```
getFlipperArpCount()

Get the number of ARP packets each traffic manager sends when an IP address is raised.

Unsigned Integer getFlipperArpCount()

getFlipperArpCountByLocation( location ) throws ObjectDoesNotExist

Get the number of ARP packets each traffic manager sends when an IP address is raised. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperArpCountByLocation(
    String location
)

getFlipperAutofailback()

Get whether Traffic IPs should automatically failback to recovered machines.

Boolean getFlipperAutofailback()

getFlipperAutofailbackByLocation( location ) throws ObjectDoesNotExist

Get whether Traffic IPs should automatically failback to recovered machines. This is a location specific function, any action will operate on the specified location.

Boolean getFlipperAutofailbackByLocation(
    String location
)

getFlipperAutofailbackDelay()

Get the delay of automatic failback after a previous failover event.

Unsigned Integer getFlipperAutofailbackDelay()

getFlipperAutofailbackDelayByLocation( location ) throws ObjectDoesNotExist

Get the delay of automatic failback after a previous failover event. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperAutofailbackDelayByLocation(
    String location
)

getFlipperChildTimeout()

Get how long (in seconds) the traffic manager should wait for status updates from any of the traffic manager’s child processes before assuming one of them is no longer servicing traffic.

Unsigned Integer getFlipperChildTimeout()

getFlipperChildTimeoutByLocation( location ) throws ObjectDoesNotExist

Get how long (in seconds) the traffic manager should wait for status updates from any of the traffic manager’s child processes before assuming one of them is no longer servicing traffic. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperChildTimeoutByLocation(
    String location
)
getFlipperFrontendCheckAddresses()
Get the IP addresses that should be used to check front-end connectivity.

String[] getFlipperFrontendCheckAddresses()

getFlipperFrontendCheckAddressesByLocation( location ) throws ObjectDoesNotExist
Get the IP addresses that should be used to check front-end connectivity. This is a location specific function, any action will operate on the specified location.

String[] getFlipperFrontendCheckAddressesByLocation(String location)

getFlipperHeartbeatMethod()
Get the method used to exchange cluster heartbeat messages.

GlobalSettings.FlipperHeartbeatMethod getFlipperHeartbeatMethod()

getFlipperHeartbeatMethodByLocation( location ) throws ObjectDoesNotExist
Get the method used to exchange cluster heartbeat messages. This is a location specific function, any action will operate on the specified location.

GlobalSettings.FlipperHeartbeatMethod getFlipperHeartbeatMethodByLocation(String location)

getFlipperIGMPInterval()
Get the interval between two unsolicited periodic IGMP Membership Report messages for Multi-Hosted Traffic IP Groups.

Unsigned Integer getFlipperIGMPInterval()

getFlipperIGMPIntervalByLocation( location ) throws ObjectDoesNotExist
Get the interval between two unsolicited periodic IGMP Membership Report messages for Multi-Hosted Traffic IP Groups. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperIGMPIntervalByLocation(String location)

getFlipperL4AccelChildTimeout()
Get how long (in seconds) the traffic manager should wait for a status update from child processes handling L4Accel services before assuming it is no longer servicing traffic.

Unsigned Integer getFlipperL4AccelChildTimeout()

getFlipperL4AccelChildTimeoutByLocation( location ) throws ObjectDoesNotExist
Get how long (in seconds) the traffic manager should wait for a status update from child processes handling L4Accel services before assuming it is no longer servicing traffic. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getFlipperL4AccelChildTimeoutByLocation(
    String location
)

**getFlipperL4AccelSyncPort()**
Get the port on which cluster members will transfer state information for L4Accel services when running in Data Plane Acceleration Mode.

Unsigned Integer getFlipperL4AccelSyncPort()

**getFlipperL4AccelSyncPortByLocation( location ) throws ObjectDoesNotExist**
Get the port on which cluster members will transfer state information for L4Accel services when running in Data Plane Acceleration Mode. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperL4AccelSyncPortByLocation(
    String location
)

**getFlipperMonitorInterval()**
Get how frequently (in milliseconds) each traffic manager checks and announces its connectivity.

Unsigned Integer getFlipperMonitorInterval()

**getFlipperMonitorIntervalByLocation( location ) throws ObjectDoesNotExist**
Get how frequently (in milliseconds) each traffic manager checks and announces its connectivity. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperMonitorIntervalByLocation(
    String location
)

**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()

**getFlipperMonitorTimeoutByLocation( location ) throws ObjectDoesNotExist**
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. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperMonitorTimeoutByLocation(
    String location
)

**getFlipperMulticastAddress()**
Get the multicast address and port used to announce connectivity (e.g. 239.100.1.1:9090).

String getFlipperMulticastAddress()
getFlipperMulticastAddressByLocation( location ) throws ObjectDoesNotExist

Get the multicast address and port used to announce connectivity (e.g. 239.100.1.1:9090). This is a location specific function, any action will operate on the specified location.

String getFlipperMulticastAddressByLocation(String location)

getFlipperUnicastPort()

Get the unicast UDP port used to announce connectivity (e.g. 9090)

Unsigned Integer getFlipperUnicastPort()

getFlipperUnicastPortByLocation( location ) throws ObjectDoesNotExist

Get the unicast UDP port used to announce connectivity (e.g. 9090). This is a location specific function, any action will operate on the specified location.

Unsigned Integer getFlipperUnicastPortByLocation(String location)

getFlipperUseBindip()

Get whether the heartbeat messages used for fault tolerance are only sent over the management network.

Boolean getFlipperUseBindip()

getFlipperUseBindipByLocation( location ) throws ObjectDoesNotExist

Get whether the heartbeat messages used for fault tolerance are only sent over the management network. This is a location specific function, any action will operate on the specified location.

Boolean getFlipperUseBindipByLocation(String location)

getFlipperVerbose()

Get whether the traffic manager should logs all the connectivity tests.

Boolean getFlipperVerbose()

getFlipperVerboseByLocation( location ) throws ObjectDoesNotExist

Get whether the traffic manager should logs all the connectivity tests. This is a location specific function, any action will operate on the specified location.

Boolean getFlipperVerboseByLocation(String location)

getGLBLoadChangeLimit()

Get the maximum change per second to load.

Unsigned Integer getGLBLoadChangeLimit()
getGLBLoadChangeLimitByLocation( location ) throws ObjectDoesNotExist

Get the maximum change per second to load. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getGLBLoadChangeLimitByLocation(
    String location
)

getGLBVerbose()

Get whether GSLB should log all DNS queries

Boolean getGLBVerbose()

getGLBVerboseByLocation( location ) throws ObjectDoesNotExist

Get whether GSLB should log all DNS queries This is a location specific function, any action will operate on the specified location.

Boolean getGLBVerboseByLocation(
    String location
)

getHistoricalTrafficDays()

Get the length of time historical traffic information is kept for, in days (0=keep indefinitely).

Unsigned Integer getHistoricalTrafficDays()

getHistoricalTrafficDaysByLocation( location ) throws ObjectDoesNotExist

Get the length of time historical traffic information is kept for, in days (0=keep indefinitely). This is a location specific function, any action will operate on the specified location.

Unsigned Integer getHistoricalTrafficDaysByLocation(
    String location
)

getIPSessionCacheSize()

Get the maximum number of entries in the IP session cache.

Unsigned Integer getIPSessionCacheSize()

getIPSessionCacheSizeByLocation( location ) throws ObjectDoesNotExist

Get the maximum number of entries in the IP session cache. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getIPSessionCacheSizeByLocation(
    String location
)

getIdleConnectionTimeout()

Get how long unused HTTP keepalive connections should be kept before being discarded, in seconds.

Unsigned Integer getIdleConnectionTimeout()
getIdleConnectionTimeoutByLocation(location) throws ObjectDoesNotExist

Get how long unused HTTP keepalive connections should be kept before being discarded, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getIdleConnectionTimeoutByLocation(
    String location
)

getJ2EESessionCacheSize()

Get the maximum number of entries in the J2EE session cache.

Unsigned Integer getJ2EESessionCacheSize()

getJ2EESessionCacheSizeByLocation(location) throws ObjectDoesNotExist

Get the maximum number of entries in the J2EE session cache. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getJ2EESessionCacheSizeByLocation(
    String location
)

getJavaClasspath()

Get extra Java CLASSPATH settings required for servlets.

String getJavaClasspath()

getJavaClasspathByLocation(location) throws ObjectDoesNotExist

Get extra Java CLASSPATH settings required for servlets. This is a location specific function, any action will operate on the specified location.

String getJavaClasspathByLocation(
    String location
)

getJavaCommand()

Get the command (and arguments) used to start Java.

String getJavaCommand()

getJavaCommandByLocation(location) throws ObjectDoesNotExist

Get the command (and arguments) used to start Java. This is a location specific function, any action will operate on the specified location.

String getJavaCommandByLocation(
    String location
)

getJavaEnabled()

Get whether to enable Java support.

Boolean getJavaEnabled()
getJavaEnabledByLocation( location ) throws ObjectDoesNotExist

Get whether to enable Java support. This is a location specific function, any action will operate on the specified location.

Boolean getJavaEnabledByLocation(
    String location
)

getJavaLib()

Get the location of the java library directory

String getJavaLib()

getJavaLibByLocation( location ) throws ObjectDoesNotExist

Get the location of the java library directory. This is a location specific function, any action will operate on the specified location.

String getJavaLibByLocation(
    String location
)

getJavaMaxConns()

Get the maximum number of Java threads

Unsigned Integer getJavaMaxConns()

getJavaMaxConnsByLocation( location ) throws ObjectDoesNotExist

Get the maximum number of Java threads. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getJavaMaxConnsByLocation(
    String location
)

ggetJavaSessionAge()

Get the default maximum age of Java session persistence

Unsigned Integer getJavaSessionAge()

getJavaSessionAgeByLocation( location ) throws ObjectDoesNotExist

Get the default maximum age of Java session persistence. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getJavaSessionAgeByLocation(
    String location
)

getKerberosVerbose()

Get whether the traffic manager should log all Kerberos activity.

Boolean getKerberosVerbose()
**getKerberosVerboseByLocation( location )** throws **ObjectDoesNotExist**

Get whether the traffic manager should log all Kerberos activity. This is a location specific function, any action will operate on the specified location.

```java
Boolean getKerberosVerboseByLocation(
    String location
)
```

**getL4AccelMaxConcurrentConnections()**

Get the maximum number of concurrent connections, in millions, that can be handled by each L4Accel child process. An appropriate amount of memory to store this many connections will be allocated when the traffic manager starts.

```java
Unsigned Integer getL4AccelMaxConcurrentConnections()
```

**getL4AccelMaxConcurrentConnectionsByLocation( location )** throws **ObjectDoesNotExist**

Get the maximum number of concurrent connections, in millions, that can be handled by each L4Accel child process. An appropriate amount of memory to store this many connections will be allocated when the traffic manager starts. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getL4AccelMaxConcurrentConnectionsByLocation(
    String location
)
```

**getLicenseServers()**

Get A list of license servers for FLA licensing.

```java
String[] getLicenseServers()
```

**getLicenseServersByLocation( location )** throws **ObjectDoesNotExist**

Get A list of license servers for FLA licensing. This is a location specific function, any action will operate on the specified location.

```java
String[] getLicenseServersByLocation(
    String location
)
```

**getListenQueueSize()**

Get the size of the listen queue for managing incoming connections.

```java
Unsigned Integer getListenQueueSize()
```

**getListenQueueSizeByLocation( location )** throws **ObjectDoesNotExist**

Get the size of the listen queue for managing incoming connections. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getListenQueueSizeByLocation(
    String location
)
getLogExportAuthHTTP()
Get the HTTP authentication method to use when exporting log entries.

```java
GlobalSettings.LogExportAuthHTTP getLogExportAuthHTTP()
```

getLogExportAuthHTTPByLocation( location ) throws ObjectDoesNotExist
Get the HTTP authentication method to use when exporting log entries. This is a location specific function, any action will operate on the specified location.

```java
GlobalSettings.LogExportAuthHTTP getLogExportAuthHTTPByLocation(
    String location
)
```

getLogExportAuthHecToken()
Get the HTTP Event Collector token to use for HTTP authentication with a Splunk server.

```java
String getLogExportAuthHecToken()
```

getLogExportAuthHecTokenByLocation( location ) throws ObjectDoesNotExist
Get the HTTP Event Collector token to use for HTTP authentication with a Splunk server. This is a location specific function, any action will operate on the specified location.

```java
String getLogExportAuthHecTokenByLocation(
    String location
)
```

getLogExportAuthUsername()
Get the username to use for HTTP basic authentication.

```java
String getLogExportAuthUsername()
```

getLogExportAuthUsernameByLocation( location ) throws ObjectDoesNotExist
Get the username to use for HTTP basic authentication. This is a location specific function, any action will operate on the specified location.

```java
String getLogExportAuthUsernameByLocation(
    String location
)
```

getLogExportEnabled()
Get whether to monitor log files and export entries to the configured endpoint.

```java
Boolean getLogExportEnabled()
```

getLogExportEnabledByLocation( location ) throws ObjectDoesNotExist
Get whether to monitor log files and export entries to the configured endpoint. This is a location specific function, any action will operate on the specified location.

```java
Boolean getLogExportEnabledByLocation(
    String location
)
```
**getLogExportEndpoint()**

Get the URL to which log entries should be sent.

```java
String getLogExportEndpoint()
```

**getLogExportEndpointByLocation( location ) throws ObjectDoesNotExist**

Get the URL to which log entries should be sent. This is a location specific function, any action will operate on the specified location.

```java
String getLogExportEndpointByLocation(
    String location
)
```

**getLogExportRequestTimeout()**

Get the number of seconds after which HTTP requests sent to the configured endpoint will be considered to have failed if no response is received.

```java
Unsigned Integer getLogExportRequestTimeout()
```

**getLogExportRequestTimeoutByLocation( location ) throws ObjectDoesNotExist**

Get the number of seconds after which HTTP requests sent to the configured endpoint will be considered to have failed if no response is received. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getLogExportRequestTimeoutByLocation(
    String location
)
```

**getLogExportTLSVerify()**

Get whether the server certificate should be verified when connecting to the endpoint. If enabled, server certificates that do not match the server name, are self-signed, have expired, have been revoked, or that are signed by an unknown CA will be rejected.

```java
Boolean getLogExportTLSVerify()
```

**getLogExportTLSVerifyByLocation( location ) throws ObjectDoesNotExist**

Get whether the server certificate should be verified when connecting to the endpoint. If enabled, server certificates that do not match the server name, are self-signed, have expired, have been revoked, or that are signed by an unknown CA will be rejected. This is a location specific function, any action will operate on the specified location.

```java
Boolean getLogExportTLSVerifyByLocation(
    String location
)
```

**getLogFlushFlushTime()**

Get the length of time to wait before flushing the request log files for each virtual server, in seconds.

```java
Unsigned Integer getLogFlushFlushTime()
```
getLogFlushFlushTimeByLocation( location ) throws ObjectDoesNotExist

Get the length of time to wait before flushing the request log files for each virtual server, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getLogFlushFlushTimeByLocation(
    String location
)

getLogInterval()

Get the length of time between log messages for log intensive features e.g. SLM, in seconds.

Unsigned Integer getLogInterval()

getLogIntervalByLocation( location ) throws ObjectDoesNotExist

Get the length of time between log messages for log intensive features e.g. SLM, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getLogIntervalByLocation(
    String location
)

getLogRate()

Get is the maximum number of connection errors logged per second.

Unsigned Integer getLogRate()

getLogRateByLocation( location ) throws ObjectDoesNotExist

Get is the maximum number of connection errors logged per second. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getLogRateByLocation(
    String location
)

getLogReopenTime()

Get the length of time to wait before re-opening request log files, to handle log file rotation, in seconds.

Unsigned Integer getLogReopenTime()

getLogReopenTimeByLocation( location ) throws ObjectDoesNotExist

Get the length of time to wait before re-opening request log files, to handle log file rotation, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getLogReopenTimeByLocation(
    String location
)

getLoginBanner()

Get the banner text to be shown on the Admin Server login page and before logging in to appliance SSH servers.

String getLoginBanner()
**getLoginDelay()**

Get the number of seconds before another login attempt can be made after a failed attempt.

Unsigned Integer getLoginDelay()

**getMaxAccepting()**

Get how many traffic manager child processes accept new connections.

Unsigned Integer getMaxAccepting()

**getMaxAcceptingByLocation( location ) throws ObjectDoesNotExist**

Get how many traffic manager child processes accept new connections. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getMaxAcceptingByLocation(
    String location
)

**getMaxIdleConnections()**

Get the maximum number of unused HTTP keepalive connections to all nodes that should maintained for re-use.

Unsigned Integer getMaxIdleConnections()

**getMaxIdleConnectionsByLocation( location ) throws ObjectDoesNotExist**

Get the maximum number of unused HTTP keepalive connections to all nodes that should maintained for re-use. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getMaxIdleConnectionsByLocation(
    String location
)

**getMaxKeepalives()**

getMaxKeepalives is deprecated, please use getMaxIdleConnections instead.

Unsigned Integer getMaxKeepalives()

**getMaxKeepalivesByLocation( location ) throws ObjectDoesNotExist**

getMaxKeepalives is deprecated, please use getMaxIdleConnections instead. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getMaxKeepalivesByLocation(
    String location
)

**getMaxLoginAttempts()**

Get the number of sequential failed login attempts that will cause a user account to be suspended. Setting this to 0 disables this feature.

Unsigned Integer getMaxLoginAttempts()
**getMaxLoginExternal()**
Get whether or not usernames blocked due to the max_login_attempts limit should also be blocked from authentication against external services (such as LDAP and RADIUS).

```java
Boolean getMaxLoginExternal()
```

**getMaxLoginSuspensionTime()**
Get number of minutes to suspend users who have exceeded the max_login_attempts limit.

```java
Unsigned Integer getMaxLoginSuspensionTime()
```

**getMaxRetries()**
This method is now obsolete and is replaced by Pool.getNodeConnectionAttempts.

```java
Unsigned Integer getMaxRetries()
```

**getMaximumFDCount()**
Get the maximum number of file descriptors that your traffic manager will allocate

```java
Unsigned Integer getMaximumFDCount()
```

**getMaximumFDCountByLocation( location ) throws ObjectDoesNotExist**
Get the maximum number of file descriptors that your traffic manager will allocate This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getMaximumFDCountByLocation(
    String location
)
```

**getMinAlphaChars()**
Get the minimum number of alphabetic characters in a password.

```java
Unsigned Integer getMinAlphaChars()
```

**getMinNumericChars()**
Get the minimum number of numeric characters in a password.

```java
Unsigned Integer getMinNumericChars()
```

**getMinPasswordLength()**
Get the minimum number of characters a password must contain.

```java
Unsigned Integer getMinPasswordLength()
```

**getMinSpecialChars()**
Get the minimum number of special characters in a password.

```java
Unsigned Integer getMinSpecialChars()
```
GlobalSettings Function Reference

**getMinUppercaseChars()**
Get the minimum number of uppercase characters in a password.

Unsigned Integer getMinUppercaseChars()

**getMonitorNumNodes()**
Get the maximum number of nodes, pools and locations that can be monitored.

Unsigned Integer getMonitorNumNodes()

**getMonitorNumNodesByLocation(location) throws ObjectDoesNotExist**
Get the maximum number of nodes, pools and locations that can be monitored. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getMonitorNumNodesByLocation(
  String location
)

**getMultipleAccept()**
Get whether your traffic manager should try and read multiple new connections each time a new client connects.

Boolean getMultipleAccept()

**getMultipleAcceptByLocation(location) throws ObjectDoesNotExist**
Get whether your traffic manager should try and read multiple new connections each time a new client connects. This is a location specific function, any action will operate on the specified location.

Boolean getMultipleAcceptByLocation(
  String location
)

**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()

**getOCSPCacheSize()**
Get the maximum number of cached client certificate OCSP results stored. This cache is used to speed up OCSP checks against client certificates by caching results.

Unsigned Integer getOCSPCacheSize()
**getOCSPCacheSizeByLocation(location) throws ObjectDoesNotExist**

Get the maximum number of cached client certificate OCSP results stored. This cache is used to speed up OCSP checks against client certificates by caching results. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getOCSPCacheSizeByLocation(String location)
```

**getOspfv2Area()**

Get the OSPF area in which the traffic manager will operate.

```java
String getOspfv2Area()
```

**getOspfv2AreaByLocation(location) throws ObjectDoesNotExist**

Get the OSPF area in which the traffic manager will operate. This is a location specific function, any action will operate on the specified location.

```java
String getOspfv2AreaByLocation(String location)
```

**getOspfv2AreaType()**

Get the type of OSPF area

```java
GlobalSettings.Ospfv2AreaType getOspfv2AreaType()
```

**getOspfv2AreaTypeByLocation(location) throws ObjectDoesNotExist**

Get the type of OSPF area. This is a location specific function, any action will operate on the specified location.

```java
GlobalSettings.Ospfv2AreaType getOspfv2AreaTypeByLocation(String location)
```

**getOspfv2AuthenticationKeyIdA()**

Get the OSPF key ID

```java
Unsigned Integer getOspfv2AuthenticationKeyIdA()
```

**getOspfv2AuthenticationKeyIdAByLocation(location) throws ObjectDoesNotExist**

Get the OSPF key ID. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getOspfv2AuthenticationKeyIdAByLocation(String location)
```

**getOspfv2AuthenticationKeyIdB()**

Get the OSPF key ID

```java
Unsigned Integer getOspfv2AuthenticationKeyIdB()
```
getOspfv2AuthenticationKeyIdBByLocation( location ) throws ObjectDoesNotExist
Get the OSPF key ID. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getOspfv2AuthenticationKeyIdBByLocation(
    String location
)

getOspfv2AuthenticationSharedSecretA()
Get the OSPF MD5 shared secret.
String getOspfv2AuthenticationSharedSecretA()

getOspfv2AuthenticationSharedSecretAByLocation( location ) throws ObjectDoesNotExist
Get the OSPF MD5 shared secret. This is a location specific function, any action will operate on the specified location.
String getOspfv2AuthenticationSharedSecretAByLocation(
    String location
)

getOspfv2AuthenticationSharedSecretB()
Get the OSPF MD5 shared secret.
String getOspfv2AuthenticationSharedSecretB()

getOspfv2AuthenticationSharedSecretBBByLocation( location ) throws ObjectDoesNotExist
Get the OSPF MD5 shared secret. This is a location specific function, any action will operate on the specified location.
String getOspfv2AuthenticationSharedSecretBBByLocation(
    String location
)

getOspfv2Enabled()
Get whether OSPF Route Health Injection is enabled.
Boolean getOspfv2Enabled()

getOspfv2EnabledByLocation( location ) throws ObjectDoesNotExist
Get whether OSPF Route Health Injection is enabled. This is a location specific function, any action will operate on the specified location.
Boolean getOspfv2EnabledByLocation(
    String location
)

getOspfv2HelloInterval()
Get the interval at which OSPF "hello" packets are sent to the network.
Unsigned Integer getOspfv2HelloInterval()
getOspfv2HelloIntervalByLocation( location ) throws ObjectDoesNotExist

Get the interval at which OSPF "hello" packets are sent to the network. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getOspfv2HelloIntervalByLocation(
    String location
)

getOspfv2RouterDeadInterval()

Get the number of seconds before declaring a silent router down.

Unsigned Integer getOspfv2RouterDeadInterval()

getOspfv2RouterDeadIntervalByLocation( location ) throws ObjectDoesNotExist

Get the number of seconds before declaring a silent router down. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getOspfv2RouterDeadIntervalByLocation(
    String location
)

getPasswordChangesPerDay()

Get the maximum number of times a password can be changed every 24 hours.

Unsigned Integer getPasswordChangesPerDay()

getPasswordReuseAfter()

Get the number of times a password must have been changed before it can be reused.

Unsigned Integer getPasswordReuseAfter()

getPostLoginBanner()

Get the banner text to be displayed on the appliance console after login.

String getPostLoginBanner()

getProtectionConncountSize()

Get the amount of shared memory reserved for an inter-process table of combined connection counts used by Service Protection classes (specified as an absolute size, eg 20MB).

String getProtectionConncountSize()

getProtectionConncountSizeByLocation( location ) throws ObjectDoesNotExist

Get the amount of shared memory reserved for an inter-process table of combined connection counts used by Service Protection classes (specified as an absolute size, eg 20MB). This is a location specific function, any action will operate on the specified location.

String getProtectionConncountSizeByLocation(
    String location
)
getRESTAuthTimeout()
Get REST authentication timeout.
Unsigned Integer getRESTAuthTimeout()

getRESTEnabled()
Get whether REST service is enabled.
Boolean getRESTEnabled()

getRESTMaxHTTPHeaderLength()
Get the maximum allowed length in bytes of a HTTP request's headers.
Unsigned Integer getRESTMaxHTTPHeaderLength()

getRESTReplicateAbsoluteTime()
Get Absolute time before configuration replication via REST.
Unsigned Integer getRESTReplicateAbsoluteTime()

getRESTReplicateLullTime()
Get Lull time for configuration replication via REST.
Unsigned Integer getRESTReplicateLullTime()

getRESTReplicateTimeout()
Get the configuration replication timeout via REST.
Unsigned Integer getRESTReplicateTimeout()

getRateClassLimit()
Get the maximum number of Rate classes allowed.
Unsigned Integer getRateClassLimit()

getRateClassLimitByLocation( location ) throws ObjectDoesNotExist
Get the maximum number of Rate classes allowed. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getRateClassLimitByLocation( String location )

getRecentConns()
Get the details of how many recently closed connections each traffic manager process should save for use with the Connections page.
Unsigned Integer getRecentConns()
Function Reference

getRecentConnsByLocation( location ) throws ObjectDoesNotExist

Get the details of how many recently closed connections each traffic manager process should save for use with the Connections page. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getRecentConnsByLocation(
    String location
)

getRecentConnsRetainTime()

Get for how long a snapshot should be retained on the Connections page.

Unsigned Integer getRecentConnsRetainTime()

getRecentConnsRetainTimeByLocation( location ) throws ObjectDoesNotExist

Get for how long a snapshot should be retained on the Connections page. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getRecentConnsRetainTimeByLocation(
    String location
)

getRecentConnsSnapshotSize()

Get the maximum number of connections each traffic manager process should show for a snapshot on the Connections page.

Unsigned Integer getRecentConnsSnapshotSize()

getRecentConnsSnapshotSizeByLocation( location ) throws ObjectDoesNotExist

Get the maximum number of connections each traffic manager process should show for a snapshot on the Connections page. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getRecentConnsSnapshotSizeByLocation(
    String location
)

getSLMClassLimit()

Get the maximum number of SLM classes allowed.

Unsigned Integer getSLMClassLimit()

getSLMClassLimitByLocation( location ) throws ObjectDoesNotExist

Get the maximum number of SLM classes allowed. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getSLMClassLimitByLocation(
    String location
)

getSNATIPLimit()

Get the maximum number of Source NAT IP addresses that can be used across all Traffic IP Groups.

Unsigned Integer getSNATIPLimit()
**getSNATIPLimitByLocation( location ) throws ObjectDoesNotExist**

Get the maximum number of Source NAT IP addresses that can be used across all Traffic IP Groups. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getSNATIPLimitByLocation(
    String location
)
```

**getSNATIPLocalPortRangeHigh()**

Get the upper boundary of the port range reserved for use by the kernel. Ports above this range will be used by the traffic manager for establishing outgoing connections.

```java
Unsigned Integer getSNATIPLocalPortRangeHigh()
```

**getSNATIPLocalPortRangeHighByLocation( location ) throws ObjectDoesNotExist**

Get the upper boundary of the port range reserved for use by the kernel. Ports above this range will be used by the traffic manager for establishing outgoing connections. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getSNATIPLocalPortRangeHighByLocation(
    String location
)
```

**getSNATSharedPoolSize()**

Get the size of the Source NAT shared memory pool used for shared storage across child processes.

```java
Unsigned Integer getSNATSharedPoolSize()
```

**getSNATSharedPoolSizeByLocation( location ) throws ObjectDoesNotExist**

Get the size of the Source NAT shared memory pool used for shared storage across child processes. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getSNATSharedPoolSizeByLocation(
    String location
)
```

**getSNMPUserCounters()**

Get the number of user defined SNMP counters (this single parameter dictates the numbers of both 32- and 64-bit user counters - there is always the same number of counters of each type).

```java
Unsigned Integer getSNMPUserCounters()
```

**getSNMPUserCountersByLocation( location ) throws ObjectDoesNotExist**

Get the number of user defined SNMP counters (this single parameter dictates the numbers of both 32- and 64-bit user counters - there is always the same number of counters of each type). This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getSNMPUserCountersByLocation(
    String location
)
```
getSSL3AllowRehandshake()
Get whether SSL / TLS re-handshakes are supported.
GlobalSettings.SSL3AllowRehandshake getSSL3AllowRehandshake()

getSSL3AllowRehandshakeByLocation( location ) throws ObjectDoesNotExist
Get whether SSL / TLS re-handshakes are supported. This is a location specific function, any action will operate on the specified location.
GlobalSettings.SSL3AllowRehandshake getSSL3AllowRehandshakeByLocation(
    String location
)

getSSL3Ciphers()
Get the list of configured SSL ciphers (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s).
String getSSL3Ciphers()

getSSL3CiphersByLocation( location ) throws ObjectDoesNotExist
Get the list of configured SSL ciphers (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s). This is a location specific function, any action will operate on the specified location.
String getSSL3CiphersByLocation(
    String location
)

getSSL3DiffieHellmanKeyLength()
Get the number of bits to use for Diffie-Hellman keys
GlobalSettings.SSL3DiffieHellmanKeyLength getSSL3DiffieHellmanKeyLength()

getSSL3DiffieHellmanKeyLengthByLocation( location ) throws ObjectDoesNotExist
Get the number of bits to use for Diffie-Hellman keys This is a location specific function, any action will operate on the specified location.
GlobalSettings.SSL3DiffieHellmanKeyLength getSSL3DiffieHellmanKeyLengthByLocation(
    String location
)

getSSL3MinRehandshakeInterval()
Get the minimum time interval (in milliseconds) between handshakes on a single SSL3/TLS connection.
Unsigned Integer getSSL3MinRehandshakeInterval()

getSSLAzureClientID()
Get the client identifier for the Azure Key Vault.
String getSSLAzureClientID()
getSSLAzureClientIDByLocation( location ) throws ObjectDoesNotExist

Get the client identifier for the Azure Key Vault. This is a location specific function, any action will operate on the specified location.

String getSSLAzureClientIDByLocation(
    String location
)

getSSLAzureVaultURL()

Get the URL of the Azure Key Vault REST API.

String getSSLAzureVaultURL()

getSSLAzureVaultURLByLocation( location ) throws ObjectDoesNotExist

Get the URL of the Azure Key Vault REST API. This is a location specific function, any action will operate on the specified location.

String getSSLAzureVaultURLByLocation(
    String location
)

getSSLAzureVerifyRESTAPICert()

Get whether the SSL certificate of the Azure Key Vault REST API will be verified using CAs present in the SSL Certificate Authorities Catalog.

Boolean getSSLAzureVerifyRESTAPICert()

getSSLAzureVerifyRESTAPICertByLocation( location ) throws ObjectDoesNotExist

Get whether the SSL certificate of the Azure Key Vault REST API will be verified using CAs present in the SSL Certificate Authorities Catalog. This is a location specific function, any action will operate on the specified location.

Boolean getSSLAzureVerifyRESTAPICertByLocation(
    String location
)

getSSLCRLMemSize()

Get the size of the CRL shared memory.

String getSSLCRLMemSize()

getSSLCRLMemSizeByLocation( location ) throws ObjectDoesNotExist

Get the size of the CRL shared memory. This is a location specific function, any action will operate on the specified location.

String getSSLCRLMemSizeByLocation(
    String location
)

getSSLDFailureCount()

getSSLDFailureCount is deprecated, please use getSSLHardwareFailureCount instead.

Unsigned Integer getSSLDFailureCount()
Function Reference

GlobalSettings

getSSLDFailureCountByLocation( location ) throws ObjectDoesNotExist

getSSLDFailureCount is deprecated, please use getSSLHardwareFailureCount instead. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getSSLDFailureCountByLocation(
    String location
)

getSSLPKCS11Lib()

getSSLPKCS11Lib is deprecated, please use getSSLHardwarePKCS11Lib instead.

String getSSLPKCS11Lib()

getSSLPKCS11LibByLocation( location ) throws ObjectDoesNotExist

getSSLPKCS11Lib is deprecated, please use getSSLHardwarePKCS11Lib instead. This is a location specific function, any action will operate on the specified location.

String getSSLPKCS11LibByLocation(
    String location
)

getSSLEllipticCurves()

Get the elliptic curve preference list for SSL connections unless overridden by virtual server or pool settings

String getSSLEllipticCurves()

getSSLEllipticCurvesByLocation( location ) throws ObjectDoesNotExist

Get the elliptic curve preference list for SSL connections unless overridden by virtual server or pool settings. This is a location specific function, any action will operate on the specified location.

String getSSLEllipticCurvesByLocation(
    String location
)

getSSLHardwareAccelerator()

Get whether your traffic manager should always attempt to use SSL hardware.

Boolean getSSLHardwareAccelerator()

getSSLHardwareAcceleratorByLocation( location ) throws ObjectDoesNotExist

Get whether your traffic manager should always attempt to use SSL hardware. This is a location specific function, any action will operate on the specified location.

Boolean getSSLHardwareAcceleratorByLocation(
    String location
)

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()
getSSLHardwareFailureCountByLocation(location) throws ObjectDoesNotExist

Get the number of consecutive failures from the SSL hardware that will be tolerated before your traffic manager tries to log in again. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getSSLHardwareFailureCountByLocation(
  String location
)

getSSLHardwarePKCS11Lib()

Get the location of the PKCS#11 library supplied by your hardware vendor.

String getSSLHardwarePKCS11Lib()

getSSLHardwarePKCS11LibByLocation(location) throws ObjectDoesNotExist

Get the location of the PKCS#11 library supplied by your hardware vendor. This is a location specific function, any action will operate on the specified location.

String getSSLHardwarePKCS11LibByLocation(
  String location
)

getSSLHardwarePKCS11SlotLabel()

Get the label of the SSL hardware slot to use.

String getSSLHardwarePKCS11SlotLabel()

getSSLHardwarePKCS11SlotLabelByLocation(location) throws ObjectDoesNotExist

Get the label of the SSL hardware slot to use. This is a location specific function, any action will operate on the specified location.

String getSSLHardwarePKCS11SlotLabelByLocation(
  String location
)

getSSLHardwarePKCS11SlotType()

Get the type of PKCS11 slot to use. Only used for PKCS11.

GlobalSettings.SSLHardwarePKCS11SlotType getSSLHardwarePKCS11SlotType()

getSSLHardwarePKCS11SlotTypeByLocation(location) throws ObjectDoesNotExist

Get the type of PKCS11 slot to use. Only used for PKCS11. This is a location specific function, any action will operate on the specified location.

GlobalSettings.SSLHardwarePKCS11SlotType getSSLHardwarePKCS11SlotTypeByLocation(
  String location
)

getSSLHardwareType()

Get the device driver library name.
GlobalSettings.SSLHardwareType getSSLHardwareType()

getSSLHardwareTypeByLocation( location ) throws ObjectDoesNotExist
Get the device driver library name. This is a location specific function, any action will operate on the
specified location.

GlobalSettings.SSLHardwareType getSSLHardwareTypeByLocation(
    String location
)

getSSLHonorFallbackSCSV()
Get whether ssl-decrypting Virtual Servers honor the Fallback SCSV

Boolean getSSLHonorFallbackSCSV()

getSSLHonorFallbackSCSVByLocation( location ) throws ObjectDoesNotExist
Get whether ssl-decrypting Virtual Servers honor the Fallback SCSV This is a location specific function, any
action will operate on the specified location.

Boolean getSSLHonorFallbackSCSVByLocation(
    String location
)

getSSLInsertExtraFragment()
Get whether SSL3 and TLS1 use one byte fragments

Boolean getSSLInsertExtraFragment()

getSSLInsertExtraFragmentByLocation( location ) throws ObjectDoesNotExist
Get whether SSL3 and TLS1 use one byte fragments This is a location specific function, any action will
operate on the specified location.

Boolean getSSLInsertExtraFragmentByLocation(
    String location
)

getSSLMaxHandshakeMessageSize()
Get the maximum acceptable size (in bytes) a SSL handshake message is permitted to be.

Unsigned Integer getSSLMaxHandshakeMessageSize()

getSSLMaxHandshakeMessageSizeByLocation( location ) throws ObjectDoesNotExist
Get the maximum acceptable size (in bytes) a SSL handshake message is permitted to be. This is a location
specific function, any action will operate on the specified location.

Unsigned Integer getSSLMaxHandshakeMessageSizeByLocation(
    String location
)
getSSLOCSPStaplingDefaultRefreshInterval()

Get how long to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling, if we don't have an up-to-date OCSP response.

Unsigned Integer getSSLOCSPStaplingDefaultRefreshInterval()

getSSLOCSPStaplingDefaultRefreshIntervalByLocation( location ) throws ObjectDoesNotExist

Get how long to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling, if we don't have an up-to-date OCSP response. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getSSLOCSPStaplingDefaultRefreshIntervalByLocation( String location )

getSSLOCSPStaplingMaximumRefreshInterval()

Get maximum number of seconds to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling. (0 means no maximum.)

 Unsigned Integer getSSLOCSPStaplingMaximumRefreshInterval()

getSSLOCSPStaplingMaximumRefreshIntervalByLocation( location ) throws ObjectDoesNotExist

Get maximum number of seconds to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling. (0 means no maximum.) This is a location specific function, any action will operate on the specified location.

Unsigned Integer getSSLOCSPStaplingMaximumRefreshIntervalByLocation( String location )

getSSLOCSPStaplingMemSize()

Get the size of the OCSP stapling response shared memory.

String getSSLOCSPStaplingMemSize()

getSSLOCSPStaplingMemSizeByLocation( location ) throws ObjectDoesNotExist

Get the size of the OCSP stapling response shared memory. This is a location specific function, any action will operate on the specified location.

String getSSLOCSPStaplingMemSizeByLocation( String location )

getSSLOCSPStaplingTimeTolerance()

Get how many seconds to allow the current time to be outside the validity time of an OCSP response before considering it invalid.

Unsigned Integer getSSLOCSPStaplingTimeTolerance()
**getSSLOCSPStaplingTimeToleranceByLocation( location ) throws ObjectDoesNotExist**

Get how many seconds to allow the current time to be outside the validity time of an OCSP response before considering it invalid. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getSSLOCSPStaplingTimeToleranceByLocation(
    String location
)
```

**getSSLOCSPStaplingVerifyResponse()**

Get whether to verify the OCSP response signature before caching a response for OCSP stapling.

```java
Boolean getSSLOCSPStaplingVerifyResponse()
```

**getSSLOCSPStaplingVerifyResponseByLocation( location ) throws ObjectDoesNotExist**

Get whether to verify the OCSP response signature before caching a response for OCSP stapling. This is a location specific function, any action will operate on the specified location.

```java
Boolean getSSLOCSPStaplingVerifyResponseByLocation(
    String location
)
```

**getSSLPreventTimingSideChannels()**

Get whether SSL3 and TLS will take performance degrading steps to prevent exposing timing side-channels.

```java
Boolean getSSLPreventTimingSideChannels()
```

**getSSLPreventTimingSideChannelsByLocation( location ) throws ObjectDoesNotExist**

Get whether SSL3 and TLS will take performance degrading steps to prevent exposing timing side-channels. This is a location specific function, any action will operate on the specified location.

```java
Boolean getSSLPreventTimingSideChannelsByLocation(
    String location
)
```

**getSSLSessionCachePerVirtualserver()**

Get whether an SSL session created by a given virtual server can only be resumed by a connection to the same virtual server.

```java
Boolean getSSLSessionCachePerVirtualserver()
```

**getSSLSessionCachePerVirtualserverByLocation( location ) throws ObjectDoesNotExist**

Get whether an SSL session created by a given virtual server can only be resumed by a connection to the same virtual server. This is a location specific function, any action will operate on the specified location.

```java
Boolean getSSLSessionCachePerVirtualserverByLocation(
    String location
)
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()

getSSLSessionCacheSizeByLocation( location ) throws ObjectDoesNotExist
Get the maximum number of entries in the SSL session cache. This is used to provide persistence based on SSL session IDs. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getSSLSessionCacheSizeByLocation(
    String location
)

getSSLSessionIDCacheExpiryTime()
Get the length of time that SSL session IDs are stored, in seconds.
Unsigned Integer getSSLSessionIDCacheExpiryTime()

getSSLSessionIDCacheExpiryTimeByLocation( location ) throws ObjectDoesNotExist
Get the length of time that SSL session IDs are stored, in seconds. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getSSLSessionIDCacheExpiryTimeByLocation(
    String location
)

getSSLSessionIDCacheSize()
Get the number of entries in the SSL session ID cache.
Unsigned Integer getSSLSessionIDCacheSize()

getSSLSessionIDCacheSizeByLocation( location ) throws ObjectDoesNotExist
Get the number of entries in the SSL session ID cache. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getSSLSessionIDCacheSizeByLocation(
    String location
)

getSSLSignatureAlgorithms()
Get the SSL signature algorithms preference list for SSL connections unless overridden by virtual server or pool settings
String getSSLSignatureAlgorithms()

getSSLSignatureAlgorithmsByLocation( location ) throws ObjectDoesNotExist
Get the SSL signature algorithms preference list for SSL connections unless overridden by virtual server or pool settings. This is a location specific function, any action will operate on the specified location.
String getSSLSignatureAlgorithmsByLocation{
getSSLSupportSSL2()
This method is now deprecated.

Boolean getSSLSupportSSL2()

getSSLSupportSSL2ByLocation( location ) throws ObjectDoesNotExist
This method is now deprecated. This is a location specific function, any action will operate on the specified location.

Boolean getSSLSupportSSL2ByLocation(
    String location
)

getSSLSupportSSL3()
Get whether SSLv3 support is enabled.

Boolean getSSLSupportSSL3()

getSSLSupportSSL3ByLocation( location ) throws ObjectDoesNotExist
Get whether SSLv3 support is enabled. This is a location specific function, any action will operate on the specified location.

Boolean getSSLSupportSSL3ByLocation(
    String location
)

getSSLSupportTLS1()
Get whether TLSv1 support is enabled.

Boolean getSSLSupportTLS1()

getSSLSupportTLS11()
Get whether TLSv1.1 support is enabled.

Boolean getSSLSupportTLS11()

getSSLSupportTLS11ByLocation( location ) throws ObjectDoesNotExist
Get whether TLSv1.1 support is enabled. This is a location specific function, any action will operate on the specified location.

Boolean getSSLSupportTLS11ByLocation(
    String location
)

getSSLSupportTLS12()
Get whether TLSv1.2 support is enabled.

Boolean getSSLSupportTLS12()
getSSLSupportTLS12ByLocation( location ) throws ObjectDoesNotExist

Get whether TLSv1.2 support is enabled. This is a location specific function, any action will operate on the specified location.

Boolean getSSLSupportTLS12ByLocation(
    String location
)

getSSLSupportTLS1ByLocation( location ) throws ObjectDoesNotExist

Get whether TLSv1 support is enabled. This is a location specific function, any action will operate on the specified location.

Boolean getSSLSupportTLS1ByLocation(
    String location
)

getSharedPoolSize()

Get is the size of shared memory pool to be used for shared storage across worker processes.

String getSharedPoolSize()

getSharedPoolSizeByLocation( location ) throws ObjectDoesNotExist

Get is the size of shared memory pool to be used for shared storage across worker processes. This is a location specific function, any action will operate on the specified location.

String getSharedPoolSizeByLocation(
    String location
)

getSoapIdleMinutes()

Get the number of minutes the SOAP server remain idle before exiting

Unsigned Integer getSoapIdleMinutes()

getSoapIdleMinutesByLocation( location ) throws ObjectDoesNotExist

Get the number of minutes the SOAP server remain idle before exiting. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getSoapIdleMinutesByLocation(
    String location
)

getSocketOptimizations()

Get whether potential network socket optimisations should be used.

GlobalSettings.SocketOptimizations getSocketOptimizations()

gSocketOptimizationsByLocation( location ) throws ObjectDoesNotExist

Get whether potential network socket optimisations should be used. This is a location specific function, any action will operate on the specified location.

GlobalSettings.SocketOptimizations getSocketOptimizationsByLocation(
    String location
)
getSsldAccel()
getSsldAccel is deprecated, please use getSSLHardwareAccelerator instead.

Boolean getSsldAccel()

getSsldAccelByLocation( location ) throws ObjectDoesNotExist
getSsldAccel is deprecated, please use getSSLHardwareAccelerator instead. This is a location specific function, any action will operate on the specified location.

Boolean getSsldAccelByLocation(String location)

getSsldLibrary()
getSsldLibrary is deprecated, please use getSSLHardwareType instead.

GlobalSettings.SsldLibrary getSsldLibrary()

getSsldLibraryByLocation( location ) throws ObjectDoesNotExist
getSsldLibrary is deprecated, please use getSSLHardwareType instead. This is a location specific function, any action will operate on the specified location.

GlobalSettings.SsldLibrary getSsldLibraryByLocation(String location)

getStateSyncTime()
Get how often the cache state is propagated to other traffic managers in the cluster, in seconds.

Unsigned Integer getStateSyncTime()

getStateSyncTimeByLocation( location ) throws ObjectDoesNotExist
Get how often the cache state is propagated to other traffic managers in the cluster, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getStateSyncTimeByLocation(String location)

getStateSyncTimeout()
Get the timeout for state propagation between cluster members, in seconds

Unsigned Integer getStateSyncTimeout()

getStateSyncTimeoutByLocation( location ) throws ObjectDoesNotExist
Get the timeout for state propagation between cluster members, in seconds. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getStateSyncTimeoutByLocation(String location)
getSystemReadBufferSize()
Get the size of the operating system’s read buffer, in bytes (0 means use the system default).
Unsigned Integer getSystemReadBufferSize()

getSystemReadBufferSizeByLocation( location ) throws ObjectDoesNotExist
Get the size of the operating system’s read buffer, in bytes (0 means use the system default). This is a location specific function, any action will operate on the specified location.
Unsigned Integer getSystemReadBufferSizeByLocation(
    String location
)

getSystemWriteBufferSize()
Get the size of the operating system’s write buffer, in bytes (0 means use the system default).
Unsigned Integer getSystemWriteBufferSize()

getSystemWriteBufferSizeByLocation( location ) throws ObjectDoesNotExist
Get the size of the operating system’s write buffer, in bytes (0 means use the system default). This is a location specific function, any action will operate on the specified location.
Unsigned Integer getSystemWriteBufferSizeByLocation(
    String location
)

getTrackUnknownUsers()
Get whether to remember past login attempts from usernames that are not known to exist (should be No for an Admin Server accessible from the public Internet).
Boolean getTrackUnknownUsers()

getTrafficIPGroupLimit()
Get the maximum number of Traffic IP Groups allowed.
Unsigned Integer getTrafficIPGroupLimit()

getTrafficIPGroupLimitByLocation( location ) throws ObjectDoesNotExist
Get the maximum number of Traffic IP Groups allowed. This is a location specific function, any action will operate on the specified location.
Unsigned Integer getTrafficIPGroupLimitByLocation(
    String location
)

getTrafficScriptExecutionTimeWarning()
Get the number of milliseconds a rule can run for before a warning is logged.
Unsigned Integer getTrafficScriptExecutionTimeWarning()
**getTrafficScriptExecutionTimeWarningByLocation( location ) throws ObjectDoesNotExist**

Get the number of milliseconds a rule can run for before a warning is logged. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getTrafficScriptExecutionTimeWarningByLocation(
    String location
)
```

**getTrafficScriptMemoryWarning()**

Get the amount of buffered network data a TrafficScript rule can buffer before a warning is logged, in bytes.

```java
Unsigned Integer getTrafficScriptMemoryWarning()
```

**getTrafficScriptMemoryWarningByLocation( location ) throws ObjectDoesNotExist**

Get the amount of buffered network data a TrafficScript rule can buffer before a warning is logged, in bytes. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getTrafficScriptMemoryWarningByLocation(
    String location
)
```

**getTrafficscriptArrayElements()**

Get the number of array elements that can be stored before additional memory is allocated.

```java
Unsigned Integer getTrafficscriptArrayElements()
```

**getTrafficscriptArrayElementsByLocation( location ) throws ObjectDoesNotExist**

Get the number of array elements that can be stored before additional memory is allocated. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getTrafficscriptArrayElementsByLocation(
    String location
)
```

**getTrafficscriptDataLocalSize()**

Get the maximum size of the TrafficScript local data pool (specified as a percentage of system RAM, e.g. ‘5%’, or an absolute size, e.g. 200MB)

```java
String getTrafficscriptDataLocalSize()
```

**getTrafficscriptDataLocalSizeByLocation( location ) throws ObjectDoesNotExist**

Get the maximum size of the TrafficScript local data pool (specified as a percentage of system RAM, e.g. ‘5%’, or an absolute size, e.g. 200MB) This is a location specific function, any action will operate on the specified location.

```java
String getTrafficscriptDataLocalSizeByLocation(
    String location
)
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()

getTrafficscriptDataSizeByLocation( location ) throws ObjectDoesNotExist  
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) This is a location specific function, any action will operate on the specified location.

String getTrafficscriptDataSizeByLocation( String location )

getTrafficscriptMaxInstr()  
Get the maximum number of instructions a TrafficScript rule will run before being aborted.

Unsigned Integer getTrafficscriptMaxInstr()

getTrafficscriptMaxInstrByLocation( location ) throws ObjectDoesNotExist  
Get the maximum number of instructions a TrafficScript rule will run before being aborted. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getTrafficscriptMaxInstrByLocation( String location )

getTrafficscriptRegexCacheSize()  
Get the number of regular expressions to cache

Unsigned Integer getTrafficscriptRegexCacheSize()

getTrafficscriptRegexCacheSizeByLocation( location ) throws ObjectDoesNotExist  
Get the number of regular expressions to cache This is a location specific function, any action will operate on the specified location.

Unsigned Integer getTrafficscriptRegexCacheSizeByLocation( String location )

getTrafficscriptRegexMatchLimit()  
Get the maximum number of ways TrafficScript will attempt to match a regular expression at each position in the subject string, before it aborts the rule and reports a TrafficScript error.

Unsigned Integer getTrafficscriptRegexMatchLimit()
Function Reference

**getTrafficscriptRegexMatchLimitByLocation(location) throws ObjectDoesNotExist**

Get the maximum number of ways TrafficScript will attempt to match a regular expression at each position in the subject string, before it aborts the rule and reports a TrafficScript error. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getTrafficscriptRegexMatchLimitByLocation(String location)

**getTrafficscriptRegexMatchWarnPerc()**

Get the percentage of trafficscript!regex_match_limit at which TrafficScript reports a performance warning.

Unsigned Integer getTrafficscriptRegexMatchWarnPerc()

**getTrafficscriptRegexMatchWarnPercByLocation(location) throws ObjectDoesNotExist**

Get the percentage of trafficscript!regex_match_limit at which TrafficScript reports a performance warning. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getTrafficscriptRegexMatchWarnPercByLocation(String location)

**getTrafficscriptVariablePoolUse()**

Get whether the 'pool.use' and 'pool.select' TrafficScript functions accept variables as well as literal strings.

Boolean getTrafficscriptVariablePoolUse()

**getTrafficscriptVariablePoolUseByLocation(location) throws ObjectDoesNotExist**

Get whether the 'pool.use' and 'pool.select' TrafficScript functions accept variables as well as literal strings. This is a location specific function, any action will operate on the specified location.

Boolean getTrafficscriptVariablePoolUseByLocation(String location)

**getTransactionExportEnabled()**

Get whether to export metadata about transactions processed by the traffic manager to an external location.

Boolean getTransactionExportEnabled()

**getTransactionExportEnabledByLocation(location) throws ObjectDoesNotExist**

Get whether to export metadata about transactions processed by the traffic manager to an external location. This is a location specific function, any action will operate on the specified location.

Boolean getTransactionExportEnabledByLocation(String location)

**getTransactionExportEndpoint()**

Get the endpoint to which transaction metadata should be exported.
String getTransactionExportEndpoint()

getTransactionExportEndpointByLocation( location ) throws ObjectDoesNotExist
Get the endpoint to which transaction metadata should be exported. This is a location specific function, any action will operate on the specified location.

String getTransactionExportEndpointByLocation(
  String location
)

getTransactionExportTLS()
Get whether the connection to the specified endpoint should be encrypted.

Boolean getTransactionExportTLS()

getTransactionExportTLSByLocation( location ) throws ObjectDoesNotExist
Get whether the connection to the specified endpoint should be encrypted. This is a location specific function, any action will operate on the specified location.

Boolean getTransactionExportTLSByLocation(
  String location
)

getTransactionExportTLSVerify()
Get whether the server certificate presented by the endpoint should be verified, preventing a connection from being established if the certificate does not match the server name, is self-signed, is expired, is revoked, or has an unknown CA.

Boolean getTransactionExportTLSVerify()

getTransactionExportTLSVerifyByLocation( location ) throws ObjectDoesNotExist
Get whether the server certificate presented by the endpoint should be verified, preventing a connection from being established if the certificate does not match the server name, is self-signed, is expired, is revoked, or has an unknown CA. This is a location specific function, any action will operate on the specified location.

Boolean getTransactionExportTLSVerifyByLocation(
  String location
)

getUipageBanner()
Get the banner text to be displayed on all Admin Server pages.

String getUipageBanner()

getUniversalSessionCacheSize()
Get the maximum number of entries in the universal session cache.

Unsigned Integer getUniversalSessionCacheSize()
**getUniversalSessionCacheSizeByLocation( location ) throws ObjectDoesNotExist**

Get the maximum number of entries in the universal session cache. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getUniversalSessionCacheSizeByLocation(
    String location
)
```

**getWebcacheAvgPathLength()**

Get the estimated average length of the path for resources to be cached

```java
Unsigned Integer getWebcacheAvgPathLength()
```

**getWebcacheAvgPathLengthByLocation( location ) throws ObjectDoesNotExist**

Get the estimated average length of the path for resources to be cached This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer getWebcacheAvgPathLengthByLocation(
    String location
)
```

**getWebcacheDisk()**

Get whether the webcache is stored on disk

```java
Boolean getWebcacheDisk()
```

**getWebcacheDiskByLocation( location ) throws ObjectDoesNotExist**

Get whether the webcache is stored on disk This is a location specific function, any action will operate on the specified location.

```java
Boolean getWebcacheDiskByLocation(
    String location
)
```

**getWebcacheDiskDir()**

Get the disk cache location

```java
String getWebcacheDiskDir()
```

**getWebcacheDiskDirByLocation( location ) throws ObjectDoesNotExist**

Get the disk cache location This is a location specific function, any action will operate on the specified location.

```java
String getWebcacheDiskDirByLocation(
    String location
)
```

**getWebcacheMaxFileNum()**

Get the maximum number of files that can be stored in the web cache

```java
Unsigned Integer getWebcacheMaxFileNum()
```
GlobalSettings Function Reference

getWebcacheMaxFileNumByLocation(location) throws ObjectDoesNotExist
Get the maximum number of files that can be stored in the web cache. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getWebcacheMaxFileNumByLocation(String location)

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()

getWebcacheMaxFileSizeByLocation(location) throws ObjectDoesNotExist
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'. This is a location specific function, any action will operate on the specified location.

String getWebcacheMaxFileSizeByLocation(String location)

getWebcacheMaxPathLength()
Get the maximum length of the path for the resource being cached

Unsigned Integer getWebcacheMaxPathLength()

getWebcacheMaxPathLengthByLocation(location) throws ObjectDoesNotExist
Get the maximum length of the path for the resource being cached. This is a location specific function, any action will operate on the specified location.

Unsigned Integer getWebcacheMaxPathLengthByLocation(String location)

getWebcacheNormalizeQuery()
Get whether the assignment sub-strings in the parameter string are put into alphabetical order.

Boolean getWebcacheNormalizeQuery()

getWebcacheNormalizeQueryByLocation(location) throws ObjectDoesNotExist
Get whether the assignment sub-strings in the parameter string are put into alphabetical order. This is a location specific function, any action will operate on the specified location.

Boolean getWebcacheNormalizeQueryByLocation(String location)

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()

**getWebcacheSizeByLocation( location ) throws ObjectDoesNotExist**
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) This is a location specific function, any action will operate on the specified location.

String getWebcacheSizeByLocation(
   String location
)

**getWebcacheVerbose()**
Get whether an X-Cache-Info header to show cacheability should be added.

Boolean getWebcacheVerbose()

**getWebcacheVerboseByLocation( location ) throws ObjectDoesNotExist**
Get whether an X-Cache-Info header to show cacheability should be added. This is a location specific function, any action will operate on the specified location.

Boolean getWebcacheVerboseByLocation(
   String location
)

**removeApplianceReturnPathRoutes( value ) throws InvalidInput, DeploymentError**
Remove a set of return path routes (MAC/IP mappings) from the configuration.

void removeApplianceReturnPathRoutes(
   GlobalSettings.ReturnPathRoute[] value
)

**removeApplianceReturnPathRoutesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**
Remove a set of return path routes (MAC/IP mappings) from the configuration. This is a location specific function, any action will operate on the specified location.

void removeApplianceReturnPathRoutesByLocation(
   String location
   GlobalSettings.ReturnPathRoute[] value
)

**removeFlipperFrontendCheckAddresses( values ) throws InvalidInput, DeploymentError**
Remove IP addresses from the list that should be used to check front-end connectivity

void removeFlipperFrontendCheckAddresses(
   String[] values
)
**GlobalSettings Function Reference**

- **removeFlipperFrontendCheckAddressesByLocation( location, values ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**
  
  Remove IP addresses from the list that should be used to check front-end connectivity. This is a location specific function, any action will operate on the specified location.

  ```java
  void removeFlipperFrontendCheckAddressesByLocation(
      String location
      String[] values
  )
  ```

- **removeLicenseServers( values )**
  
  Remove a list of license servers for FLA licensing.

  ```java
  void removeLicenseServers(
      String[] values
  )
  ```

- **removeLicenseServersByLocation( location, values ) throws ObjectDoesNotExist**
  
  Remove a list of license servers for FLA licensing. This is a location specific function, any action will operate on the specified location.

  ```java
  void removeLicenseServersByLocation(
      String location
      String[] values
  )
  ```

- **setASPSessionCacheSize( value ) throws InvalidInput, DeploymentError**
  
  Set the maximum number of entries in the ASP session cache.

  ```java
  void setASPSessionCacheSize(
      Unsigned Integer value
  )
  ```

- **setASPSessionCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**
  
  Set the maximum number of entries in the ASP session cache. This is a location specific function, any action will operate on the specified location.

  ```java
  void setASPSessionCacheSizeByLocation(
      String location
      Unsigned Integer value
  )
  ```

- **setAcceptingDelay( value ) throws InvalidInput, DeploymentError**
  
  Set how often each traffic manager child process checks whether it should be accepting new connections.

  ```java
  void setAcceptingDelay(
      Unsigned Integer value
  )
  ```
setAcceptingDelayByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how often each traffic manager child process checks whether it should be accepting new connections. This is a location specific function, any action will operate on the specified location.

```java
void setAcceptingDelayByLocation(
    String location
    Unsigned Integer value
)
```

setAdminAllowRehandshake( value ) throws InvalidInput, DeploymentError

Set whether SSL / TLS re-handshakes are supported.

```java
void setAdminAllowRehandshake(
    GlobalSettings.AdminAllowRehandshake value
)
```

setAdminDiffieHellmanKeyLength( value ) throws InvalidInput, DeploymentError

Set the number of bits to use for Diffie-Hellman keys

```java
void setAdminDiffieHellmanKeyLength(
    GlobalSettings.AdminDiffieHellmanKeyLength value
)
```

setAdminHonorFallbackSCSV( value ) throws InvalidInput, DeploymentError

Set whether admin server, internal control port and config daemon honor the Fallback SCSV

```java
void setAdminHonorFallbackSCSV(
    Boolean value
)
```

setAdminInsertExtraFragment( value ) throws InvalidInput, DeploymentError

Set whether admin server SSL3 and TLS1 use one byte fragments

```java
void setAdminInsertExtraFragment(
    Boolean value
)
```

setAdminMinRehandshakeInterval( value ) throws InvalidInput, DeploymentError

Set the minimum time interval (in milliseconds) between handshakes on a single SSL3/TLS connection.

```java
void setAdminMinRehandshakeInterval(
    Unsigned Integer value
)
```

setAdminSSLCiphers( value ) throws InvalidInput, DeploymentError

Set the list of configured SSL ciphers for admin server and internal connections (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s).

```java
void setAdminSSLCiphers(
    String value
)
```
**setAdminSSLEllipticCurves( value ) throws InvalidInput, DeploymentError**

Set the elliptic curve preference list for SSL connections to the admin server and within the traffic manager cluster.

```java
void setAdminSSLEllipticCurves(
    String value
)
```

**setAdminSSLMaxHandshakeMessageSize( value ) throws InvalidInput, DeploymentError**

Set the maximum acceptable size (in bytes) a SSL handshake message is permitted to be for admin and internal connections.

```java
void setAdminSSLMaxHandshakeMessageSize(
    Unsigned Integer value
)
```

**setAdminSSLPreventTimingSideChannels( value ) throws InvalidInput, DeploymentError**

Set whether SSLv3 and TLS used by the admin server and internal connections will take performance degrading steps to prevent exposing timing side-channels.

```java
void setAdminSSLPreventTimingSideChannels(
    Boolean value
)
```

**setAdminSSLSignatureAlgorithms( value ) throws InvalidInput, DeploymentError**

Set the SSL signature algorithms preference list for SSL connections to the admin server and within the zxtm cluster.

```java
void setAdminSSLSignatureAlgorithms(
    String value
)
```

**setAdminSSLSupportTLS11( value ) throws InvalidInput, DeploymentError**

Set whether TLSv1.1 support is enabled for admin server and internal connections.

```java
void setAdminSSLSupportTLS11(
    Boolean value
)
```

**setAdminSSLSupportTLS12( value ) throws InvalidInput, DeploymentError**

Set whether TLSv1.2 support is enabled for admin server and internal connections.

```java
void setAdminSSLSupportTLS12(
    Boolean value
)
```

**setAdminSupportSSL2( value ) throws InvalidInput, DeploymentError**

This method is now deprecated.

```java
void setAdminSupportSSL2(
    Boolean value
)
```
setAdminSupportSSL3( value ) throws InvalidInput, DeploymentError
Set whether SSLv3 support is enabled for admin server and internal connections.

void setAdminSupportSSL3(
    Boolean value
)

setAdminSupportTLS1( value ) throws InvalidInput, DeploymentError
Set whether TLSv1 support is enabled for admin server and internal connections.

void setAdminSupportTLS1(
    Boolean value
)

setAfmEnabled( value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist, LicenseError
Set whether the Application Firewall is enabled

void setAfmEnabled(
    Boolean 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
)

setAlertEmailIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the length of time between alert emails, in seconds. Several alert messages will be stored up and sent in one email. This is a location specific function, any action will operate on the specified location.

void setAlertEmailIntervalByLocation(
    String location
    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
)

setAlertEmailMaxAttemptsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the number of times to attempt sending an email before giving up. This is a location specific function, any action will operate on the specified location.

void setAlertEmailMaxAttemptsByLocation(

GlobalSettings Function Reference

String location
Unsigned Integer value
)

setAllowConsecutiveChars( value ) throws InvalidInput, DeploymentError
Set whether the same character can appear consecutively in passwords.

void setAllowConsecutiveChars(
  Boolean value
)

setApplianceReturnPathRoutes( value ) throws InvalidInput, DeploymentError
Replace the configuration with the specified set of return path routes (MAC/IP mappings).

void setApplianceReturnPathRoutes(
  GlobalSettings.ReturnPathRoute[] value
)

setApplianceReturnPathRoutesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Replace the configuration with the specified set of return path routes (MAC/IP mappings). This is a location specific function, any action will operate on the specified location.

void setApplianceReturnPathRoutesByLocation(
  String location,
  GlobalSettings.ReturnPathRoute[] value
)

setApplianceReturnPathRoutingEnabled( value ) throws InvalidInput, DeploymentError
Set whether return path routing is enabled.

void setApplianceReturnPathRoutingEnabled(
  Boolean value
)

setApplianceReturnPathRoutingEnabledByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set whether return path routing is enabled. This is a location specific function, any action will operate on the specified location.

void setApplianceReturnPathRoutingEnabledByLocation(
  String location
  Boolean value
)

setAptimizerMaxDependentFetchSize( value ) throws InvalidInput, DeploymentError
Set the maximum size of a dependent resource that can be sent to Web Accelerator. Set to 0 to disable limit.

void setAptimizerMaxDependentFetchSize(
  String value
)
setAptimizerMaxOriginalContentBufferSize(value) throws InvalidInput, DeploymentError

Set the maximum size of original content buffer for content sent to Web Accelerator.

void setAptimizerMaxOriginalContentBufferSize(String value)

setAptimizerWatchdogInterval(value) throws InvalidInput, DeploymentError

Set how long (in seconds) the Web Accelerator watchdog mechanism should keep count of crashes for.

void setAptimizerWatchdogInterval(Unsigned Integer value)

setAptimizerWatchdogLimit(value) throws InvalidInput, DeploymentError

Set the maximum number of times the Web Accelerator sub-process will be restarted.

void setAptimizerWatchdogLimit(Unsigned Integer value)

setAuditlogViaEventd(value) throws InvalidInput, DeploymentError

Set whether the auditlog is to be mirrored to EventD.

void setAuditlogViaEventd(Boolean value)

setAuditlogViaEventdByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the auditlog is to be mirrored to EventD. This is a location specific function, any action will operate on the specified location.

void setAuditlogViaEventdByLocation(String location, Boolean value)

setAuditlogViaSyslog(value) throws InvalidInput, DeploymentError

Set whether the auditlog is to be mirrored to the syslog

void setAuditlogViaSyslog(Boolean value)

setAuditlogViaSyslogByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the auditlog is to be mirrored to the syslog. This is a location specific function, any action will operate on the specified location.

void setAuditlogViaSyslogByLocation(String location, Boolean value)
setAutoscalerVerbose( value ) throws InvalidInput, DeploymentError
Set detailed logging of autoscaler status and actions

```java
void setAutoscalerVerbose(
    Boolean value
)
```

setAutoscalerVerboseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set detailed logging of autoscaler status and actions. This is a location specific function, any action will operate on the specified location.

```java
void setAutoscalerVerboseByLocation(
    String location
    Boolean value
)
```

setBackendKeepaliveTimeout( value ) throws InvalidInput, DeploymentError
setBackendKeepaliveTimeout is deprecated, please use setIdleConnectionTimeout instead.

```java
void setBackendKeepaliveTimeout(
    Unsigned Integer value
)
```

setBackendKeepaliveTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
setBackendKeepaliveTimeout is deprecated, please use setIdleConnectionTimeout instead. This is a location specific function, any action will operate on the specified location.

```java
void setBackendKeepaliveTimeoutByLocation(
    String location
    Unsigned Integer value
)
```

setBandwidthSharing( value ) throws InvalidInput, DeploymentError
This method is now obsolete and is replaced by Catalog.Bandwidth.setSharing.

```java
void setBandwidthSharing(
    Boolean value
)
```

setBannerAccept( value ) throws InvalidInput, DeploymentError
Set whether or not users must explicitly agree to the displayed login_banner text before logging in to the Admin Server.

```java
void setBannerAccept(
    Boolean value
)
```

setBgpAsNumber( value ) throws InvalidInput, DeploymentError
Set the number of the BGP AS in which the traffic manager will operate.
Function Reference

GlobalSettings

```java
void setBgpAsNumber(
    Unsigned Integer value
)

setBgpAsNumberByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of the BGP AS in which the traffic manager will operate. This is a location specific function, any action will operate on the specified location.

void setBgpAsNumberByLocation(
    String location
    Unsigned Integer value
)

setBgpEnabled( value ) throws InvalidInput, DeploymentError

Set whether BGP Route Health Injection is enabled.

void setBgpEnabled(
    Boolean value
)

setBgpEnabledByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether BGP Route Health Injection is enabled. This is a location specific function, any action will operate on the specified location.

void setBgpEnabledByLocation(
    String location
    Boolean value
)

setBootloaderPassword( password ) throws InvalidInput, DeploymentError

Set the bootloader password.

void setBootloaderPassword(
    String password
)

setChunkSize( value ) throws InvalidInput, DeploymentError

Set the default chunk size for reading and writing data, in bytes.

void setChunkSize(
    Unsigned Integer value
)

setChunkSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the default chunk size for reading and writing data, in bytes. This is a location specific function, any action will operate on the specified location.

void setChunkSizeByLocation(
    String location
    Unsigned Integer value
)
GlobalSettings Function Reference

**setClientFirstOpt** (value) throws InvalidInput, DeploymentError

Set whether client-first network socket optimisations should be used.

```java
void setClientFirstOpt(
    Boolean value
)
```

**setClientFirstOptByLocation** (location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether client-first network socket optimisations should be used. This is a location specific function, any action will operate on the specified location.

```java
void setClientFirstOptByLocation(
    String location
    Boolean value
)
```

**setControlAllowHosts** (value) throws InvalidInput, DeploymentError

Set the hosts that are allowed to contact the internal administration port on each traffic manager.

```java
void setControlAllowHosts(
    String value
)
```

**setControlAllowHostsByLocation** (location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the hosts that are allowed to contact the internal administration port on each traffic manager. This is a location specific function, any action will operate on the specified location.

```java
void setControlAllowHostsByLocation(
    String location
    String value
)
```

**setControlCanUpdateDefault** (value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the value of the control!canupdate key for new cluster members.

```java
void setControlCanUpdateDefault(
    Boolean value
)
```

**setDNSCacheExpiryTime** (value) throws InvalidInput, DeploymentError

This method is now deprecated and is replaced by setDNSCacheMaxTTL/setDNSCacheMinTTL.

```java
void setDNSCacheExpiryTime(
    Unsigned Integer value
)
```

**setDNSCacheExpiryTimeByLocation** (location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

This method is now deprecated and is replaced by setDNSCacheMaxTTL/setDNSCacheMinTTL. This is a location specific function, any action will operate on the specified location.
void setDNSCacheExpiryTimeByLocation(
    String location
    Unsigned Integer value
)

setDNSCacheMaxTTL( value ) throws InvalidInput, DeploymentError
Set the maximum time entries are stored in the DNS cache for, in seconds.
void setDNSCacheMaxTTL(
    Unsigned Integer value
)

setDNSCacheMaxTTLByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum time entries are stored in the DNS cache for, in seconds. This is a location specific function, any action will operate on the specified location.
void setDNSCacheMaxTTLByLocation(
    String location
    Unsigned Integer value
)

setDNSCacheMinTTL( value ) throws InvalidInput, DeploymentError
Set the minimum time entries are stored in the DNS cache for, in seconds.
void setDNSCacheMinTTL(
    Unsigned Integer value
)

setDNSCacheMinTTLByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the minimum time entries are stored in the DNS cache for, in seconds. This is a location specific function, any action will operate on the specified location.
void setDNSCacheMinTTLByLocation(
    String location
    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
)

setDNSCacheNegativeExpiryTimeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the time failed lookups are stored in the DNS cache for, in seconds. This is a location specific function, any action will operate on the specified location.
void setDNSCacheNegativeExpiryTimeByLocation(
    String location
    Unsigned Integer value
)
setDNSCacheSize( value ) throws InvalidInput, DeploymentError
Set the maximum number of entries in the DNS cache.

```java
void setDNSCacheSize(
    Unsigned Integer value
)
```

setDNSCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum number of entries in the DNS cache. This is a location specific function, any action will operate on the specified location.

```java
void setDNSCacheSizeByLocation(
    String location
    Unsigned Integer value
)
```

setDNSTimeout( value ) throws InvalidInput, DeploymentError
Set the timeout for receiving a response from a DNS Server, in seconds.

```java
void setDNSTimeout(
    Unsigned Integer value
)
```

setDNSTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the timeout for receiving a response from a DNS Server, in seconds. This is a location specific function, any action will operate on the specified location.

```java
void setDNSTimeoutByLocation(
    String location
    Unsigned Integer value
)
```

setDataPlaneAccelerationCores( value ) throws InvalidInput, DeploymentError
Set the number of CPU cores assigned to assist with data plane acceleration. These cores are dedicated to reading and writing packets to the network interface cards and distributing packets between the traffic manager processes.

```java
void setDataPlaneAccelerationCores(
    GlobalSettings.DataPlaneAccelerationCores value
)
```

setDataPlaneAccelerationCoresByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the number of CPU cores assigned to assist with data plane acceleration. These cores are dedicated to reading and writing packets to the network interface cards and distributing packets between the traffic manager processes. This is a location specific function, any action will operate on the specified location.

```java
void setDataPlaneAccelerationCoresByLocation(
    String location
    GlobalSettings.DataPlaneAccelerationCores value
)
setDataPlaneAccelerationMode( value ) throws InvalidInput, DeploymentError
Set whether Data Plane Acceleration Mode is enabled.

```java
void setDataPlaneAccelerationMode(
    Boolean value
)
```

setDataPlaneAccelerationModeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set whether Data Plane Acceleration Mode is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setDataPlaneAccelerationModeByLocation(
    String location
    Boolean value
)
```

setDataPlaneAccelerationTCPDelayAck( value ) throws InvalidInput, DeploymentError
Set the time, in milliseconds, to delay sending a TCP ACK response, providing an opportunity for additional data to be incorporated into the response and potentially improving network performance. The setting affects TCP connections handled by layer 7 services running in Data Plane Acceleration mode.

```java
void setDataPlaneAccelerationTCPDelayAck(
    Unsigned Integer value
)
```

setDataPlaneAccelerationTCPDelayAckByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the time, in milliseconds, to delay sending a TCP ACK response, providing an opportunity for additional data to be incorporated into the response and potentially improving network performance. The setting affects TCP connections handled by layer 7 services running in Data Plane Acceleration mode. This is a location specific function, any action will operate on the specified location.

```java
void setDataPlaneAccelerationTCPDelayAckByLocation(
    String location
    Unsigned Integer value
)
```

setDataPlaneAccelerationTCPWinScale( value ) throws InvalidInput, DeploymentError
Set the TCP window scale option, which configures the size of the receive window for TCP connections handled by layer 7 services when running in Data Plane Acceleration mode.

```java
void setDataPlaneAccelerationTCPWinScale(
    Unsigned Integer value
)
```
setDataPlaneAccelerationTCPWinScaleByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the TCP window scale option, which configures the size of the receive window for TCP connections handled by layer 7 services when running in Data Plane Acceleration mode. This is a location specific function, any action will operate on the specified location.

```java
void setDataPlaneAccelerationTCPWinScaleByLocation(
    String location
    Unsigned Integer value
)
```

setDeadTime( value ) throws InvalidInput, DeploymentError

This method is now obsolete and is replaced by Pool.setNodeFailTime.

```java
void setDeadTime(
    Unsigned Integer value
)
```

setEC2AccessKeyID( value ) throws InvalidInput, DeploymentError

Set the Access Key ID used for interacting with the EC2 API.

```java
void setEC2AccessKeyID(
    String value
)
```

setEC2AccessKeyIDByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the Access Key ID used for interacting with the EC2 API. This is a location specific function, any action will operate on the specified location.

```java
void setEC2AccessKeyIDByLocation(
    String location
    String value
)
```

setEC2AwstoolTimeout( value ) throws InvalidInput, DeploymentError

Set the timeout for awstool requests to the AWS Query Server

```java
void setEC2AwstoolTimeout(
    Unsigned Integer value
)
```

setEC2AwstoolTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the timeout for awstool requests to the AWS Query Server. This is a location specific function, any action will operate on the specified location.

```java
void setEC2AwstoolTimeoutByLocation(
    String location
    Unsigned Integer value
)
```


**setEC2Endpoint( value ) throws InvalidInput, DeploymentError**

Set URL for the Amazon EC2 AWS endpoint.

```java
void setEC2Endpoint(String value)
```

**setEC2EndpointByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set URL for the Amazon EC2 AWS endpoint. This is a location specific function, any action will operate on the specified location.

```java
void setEC2EndpointByLocation(String location, String value)
```

**setEC2MetadataServer( value ) throws InvalidInput, DeploymentError**

Set URL for the EC2 metadata server.

```java
void setEC2MetadataServer(String value)
```

**setEC2MetadataServerByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set URL for the EC2 metadata server. This is a location specific function, any action will operate on the specified location.

```java
void setEC2MetadataServerByLocation(String location, String value)
```

**setEC2SecretAccessKey( value ) throws InvalidInput, DeploymentError**

Set the Secret Access Key used for interacting with the EC2 API.

```java
void setEC2SecretAccessKey(String value)
```

**setEC2SecretAccessKeyByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the Secret Access Key used for interacting with the EC2 API. This is a location specific function, any action will operate on the specified location.

```java
void setEC2SecretAccessKeyByLocation(String location, String value)
```
**setEC2VerifyEndpointCert( value ) throws InvalidInput, DeploymentError**

Set Whether to verify Amazon EC2 endpoint's certificate using CAs present in SSL Certificate Authorities Catalog.

```java
void setEC2VerifyEndpointCert(
    Boolean value
)
```

**setEC2VerifyEndpointCertByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set Whether to verify Amazon EC2 endpoint's certificate using CAs present in SSL Certificate Authorities Catalog. This is a location specific function, any action will operate on the specified location.

```java
void setEC2VerifyEndpointCertByLocation(
    String location
    Boolean value
)
```

**setErrorLevel( value ) throws InvalidInput, DeploymentError**

This method is now deprecated.

```java
void setErrorLevel(
    GlobalSettings.ErrorLevel value
)
```

**setErrorLevelByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

This method is now deprecated. This is a location specific function, any action will operate on the specified location.

```java
void setErrorLevelByLocation(
    String location
    GlobalSettings.ErrorLevel value
)
```

**setErrorLogFile( value ) throws InvalidInput, DeploymentError**

Set the filename that errors are logged to.

```java
void setErrorLogFile(
    String value
)
```

**setErrorLogFileByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the filename that errors are logged to. This is a location specific function, any action will operate on the specified location.

```java
void setErrorLogFileByLocation(
    String location
    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.

```java
void setFTPDataBindLow(
    Boolean value
)
```

setFTPDataBindLowByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

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. This is a location specific function, any action will operate on the specified location.

```java
void setFTPDataBindLowByLocation(
    String location,
    Boolean value
)
```

setFipsEnabled( value ) throws InvalidInput, DeploymentError, InvalidOperation

Set whether FIPS Mode is enabled.

```java
void setFipsEnabled(
    Boolean value
)
```

setFlipperArpCount( value ) throws InvalidInput, DeploymentError

Set the number of ARP packets each traffic manager sends when an IP address is raised.

```java
void setFlipperArpCount(
    Unsigned Integer value
)
```

setFlipperArpCountByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of ARP packets each traffic manager sends when an IP address is raised. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperArpCountByLocation(
    String location,
    Unsigned Integer value
)
```

setFlipperAutofailback( value ) throws InvalidInput, DeploymentError

Set whether Traffic IPs should automatically failback to recovered machines.

```java
void setFlipperAutofailback(
    Boolean value
)
```
setFlipperAutofailbackByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether Traffic IPs should automatically failback to recovered machines. This is a location specific function, any action will operate on the specified location.

void setFlipperAutofailbackByLocation{
    String location
    Boolean value
}

setFlipperAutofailbackDelay( value ) throws InvalidInput, DeploymentError

Set the delay of automatic failback after a previous failover event.

void setFlipperAutofailbackDelay{
    Unsigned Integer value
}

setFlipperAutofailbackDelayByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the delay of automatic failback after a previous failover event. This is a location specific function, any action will operate on the specified location.

void setFlipperAutofailbackDelayByLocation{
    String location
    Unsigned Integer value
}

setFlipperChildTimeout( value ) throws InvalidInput, DeploymentError

Set how long (in seconds) the traffic manager should wait for status updates from any of the traffic manager’s child processes before assuming one of them is no longer servicing traffic.

void setFlipperChildTimeout{
    Unsigned Integer value
}

setFlipperChildTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how long (in seconds) the traffic manager should wait for status updates from any of the traffic manager’s child processes before assuming one of them is no longer servicing traffic. This is a location specific function, any action will operate on the specified location.

void setFlipperChildTimeoutByLocation{
    String location
    Unsigned Integer value
}

setFlipperFrontendCheckAddresses( values ) throws InvalidInput, DeploymentError

Set the IP addresses that should be used to check front-end connectivity.

void setFlipperFrontendCheckAddresses{
    String[] values
}
setFlipperFrontendCheckAddressesByLocation( location, values ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the IP addresses that should be used to check front-end connectivity. This is a location specific function, any action will operate on the specified location.

void setFlipperFrontendCheckAddressesByLocation(
    String location
    String[] values
)

setFlipperHeartbeatMethod( value ) throws InvalidInput, DeploymentError

Set the method used to exchange cluster heartbeat messages.

void setFlipperHeartbeatMethod(
    GlobalSettings.FlipperHeartbeatMethod value
)

setFlipperHeartbeatMethodByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the method used to exchange cluster heartbeat messages. This is a location specific function, any action will operate on the specified location.

void setFlipperHeartbeatMethodByLocation(
    String location
    GlobalSettings.FlipperHeartbeatMethod value
)

setFlipperIGMPInterval( value ) throws InvalidInput, DeploymentError

Set the interval between two unsolicited periodic IGMP Membership Report messages for Multi-Hosted Traffic IP Groups.

void setFlipperIGMPInterval(
    Unsigned Integer value
)

setFlipperIGMPIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the interval between two unsolicited periodic IGMP Membership Report messages for Multi-Hosted Traffic IP Groups. This is a location specific function, any action will operate on the specified location.

void setFlipperIGMPIntervalByLocation(
    String location
    Unsigned Integer value
)

setFlipperL4AccelChildTimeout( value ) throws InvalidInput, DeploymentError

Set how long (in seconds) the traffic manager should wait for a status update from child processes handling L4Accel services before assuming it is no longer servicing traffic.

void setFlipperL4AccelChildTimeout(
    Unsigned Integer value
)
setFlipperL4AccelChildTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how long (in seconds) the traffic manager should wait for a status update from child processes handling L4Accel services before assuming it is no longer servicing traffic. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperL4AccelChildTimeoutByLocation(
    String location
    Unsigned Integer value
)
```

setFlipperL4AccelSyncPort( value ) throws InvalidInput, DeploymentError

Set the port on which cluster members will transfer state information for L4Accel services when running in Data Plane Acceleration Mode.

```java
void setFlipperL4AccelSyncPort(
    Unsigned Integer value
)
```

setFlipperL4AccelSyncPortByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the port on which cluster members will transfer state information for L4Accel services when running in Data Plane Acceleration Mode. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperL4AccelSyncPortByLocation(
    String location
    Unsigned Integer value
)
```

setFlipperMonitorInterval( value ) throws InvalidInput, DeploymentError

Set how frequently (in milliseconds) each traffic manager checks and announces its connectivity.

```java
void setFlipperMonitorInterval(
    Unsigned Integer value
)
```

setFlipperMonitorIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how frequently (in milliseconds) each traffic manager checks and announces its connectivity. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperMonitorIntervalByLocation(
    String location
    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.

```java
void setFlipperMonitorTimeout(
    Unsigned Integer value
)
```
setFlipperMonitorTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

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. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperMonitorTimeoutByLocation(
    String location
    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).

```java
void setFlipperMulticastAddress(
    String value
)
```

setFlipperMulticastAddressByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the multicast address and port used to announce connectivity (e.g. 239.100.1.1:9090). This is a location specific function, any action will operate on the specified location.

```java
void setFlipperMulticastAddressByLocation(
    String location
    String value
)
```

setFlipperUnicastPort( value ) throws InvalidInput, DeploymentError

Set the unicast UDP port used to announce connectivity (e.g. 9090)

```java
void setFlipperUnicastPort(
    Unsigned Integer value
)
```

setFlipperUnicastPortByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the unicast UDP port used to announce connectivity (e.g. 9090) This is a location specific function, any action will operate on the specified location.

```java
void setFlipperUnicastPortByLocation(
    String location
    Unsigned Integer value
)
```

setFlipperUseBindip( value ) throws InvalidInput, DeploymentError

Set whether the heartbeat messages used for fault tolerance are only sent over the management network.

```java
void setFlipperUseBindip(
    Boolean value
)
```
GlobalSettings Function Reference

setFlipperUseBindipByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the heartbeat messages used for fault tolerance are only sent over the management network. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperUseBindipByLocation(
    String location
    Boolean value
)
```

setFlipperVerbose( value ) throws InvalidInput, DeploymentError

Set whether the traffic manager should logs all the connectivity tests.

```java
void setFlipperVerbose(
    Boolean value
)
```

setFlipperVerboseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the traffic manager should logs all the connectivity tests. This is a location specific function, any action will operate on the specified location.

```java
void setFlipperVerboseByLocation(
    String location
    Boolean value
)
```

setGLBLoadChangeLimit( value ) throws InvalidInput, DeploymentError

Set the maximum change per second to load.

```java
void setGLBLoadChangeLimit(
    Unsigned Integer value
)
```

setGLBLoadChangeLimitByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum change per second to load. This is a location specific function, any action will operate on the specified location.

```java
void setGLBLoadChangeLimitByLocation(
    String location
    Unsigned Integer value
)
```

setGLBVerbose( value ) throws InvalidInput, DeploymentError

Set whether GSLB should log all DNS queries

```java
void setGLBVerbose(
    Boolean value
)
```
setGLBVerboseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether GSLB should log all DNS queries. This is a location specific function, any action will operate on the specified location.

```java
void setGLBVerboseByLocation(
    String location
    Boolean value
)
```

setHistoricalTrafficDays( value ) throws InvalidInput, DeploymentError

Set the length of time historical traffic information is kept for, in days (0=keep indefinitely).

```java
void setHistoricalTrafficDays(
    Unsigned Integer value
)
```

setHistoricalTrafficDaysByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the length of time historical traffic information is kept for, in days (0=keep indefinitely). This is a location specific function, any action will operate on the specified location.

```java
void setHistoricalTrafficDaysByLocation(
    String location
    Unsigned Integer value
)
```

setIPSessionCacheSize( value ) throws InvalidInput, DeploymentError

Set the maximum number of entries in the IP session cache.

```java
void setIPSessionCacheSize(
    Unsigned Integer value
)
```

setIPSessionCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of entries in the IP session cache. This is a location specific function, any action will operate on the specified location.

```java
void setIPSessionCacheSizeByLocation(
    String location
    Unsigned Integer value
)
```

setIdleConnectionTimeout( value ) throws InvalidInput, DeploymentError

Set how long unused HTTP keepalive connections should be kept before being discarded, in seconds.

```java
void setIdleConnectionTimeout(
    Unsigned Integer value
)
```
setIdleConnectionTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how long unused HTTP keepalive connections should be kept before being discarded, in seconds. This is a location specific function, any action will operate on the specified location.

```java
void setIdleConnectionTimeoutByLocation(
    String location
    Unsigned Integer value
)
```

setJ2EESessionCacheSize( value ) throws InvalidInput, DeploymentError

Set the maximum number of entries in the J2EE session cache.

```java
void setJ2EESessionCacheSize(
    Unsigned Integer value
)
```

setJ2EESessionCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of entries in the J2EE session cache. This is a location specific function, any action will operate on the specified location.

```java
void setJ2EESessionCacheSizeByLocation(
    String location
    Unsigned Integer value
)
```

setJavaClasspath( value ) throws InvalidInput, DeploymentError

Set extra Java CLASSPATH settings required for servlets.

```java
void setJavaClasspath(
    String value
)
```

setJavaClasspathByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set extra Java CLASSPATH settings required for servlets. This is a location specific function, any action will operate on the specified location.

```java
void setJavaClasspathByLocation(
    String location
    String value
)
```

setJavaCommand( value ) throws InvalidInput, DeploymentError

Set the command (and arguments) used to start Java.

```java
void setJavaCommand(
    String value
)
```
setJavaCommandByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the command (and arguments) used to start Java. This is a location specific function, any action will operate on the specified location.

```java
void setJavaCommandByLocation(
    String location
    String value
)
```

setJavaEnabled( value ) throws InvalidInput, DeploymentError

Set whether to enable Java support.

```java
void setJavaEnabled(
    Boolean value
)
```

setJavaEnabledByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether to enable Java support. This is a location specific function, any action will operate on the specified location.

```java
void setJavaEnabledByLocation(
    String location
    Boolean value
)
```

setJavaLib( value ) throws InvalidInput, DeploymentError

Set the location of the java library directory

```java
void setJavaLib(
    String value
)
```

setJavaLibByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the location of the java library directory This is a location specific function, any action will operate on the specified location.

```java
void setJavaLibByLocation(
    String location
    String value
)
```

setJavaMaxConns( value ) throws InvalidInput, DeploymentError

Set the maximum number of Java threads

```java
void setJavaMaxConns(
    Unsigned Integer value
)
```
setJavaMaxConnsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of Java threads. This is a location specific function, any action will operate on the specified location.

```java
void setJavaMaxConnsByLocation(
    String location
    Unsigned Integer value
)
```

setJavaSessionAge( value ) throws InvalidInput, DeploymentError

Set the default maximum age of Java session persistence

```java
void setJavaSessionAge(
    Unsigned Integer value
)
```

setJavaSessionAgeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the default maximum age of Java session persistence. This is a location specific function, any action will operate on the specified location.

```java
void setJavaSessionAgeByLocation(
    String location
    Unsigned Integer value
)
```

setKerberosVerbose( value ) throws InvalidInput, DeploymentError

Set whether the traffic manager should log all Kerberos activity.

```java
void setKerberosVerbose(
    Boolean value
)
```

setKerberosVerboseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the traffic manager should log all Kerberos activity. This is a location specific function, any action will operate on the specified location.

```java
void setKerberosVerboseByLocation(
    String location
    Boolean value
)
```

setL4AccelMaxConcurrentConnections( value ) throws InvalidInput, DeploymentError

Set the maximum number of concurrent connections, in millions, that can be handled by each L4Accel child process. An appropriate amount of memory to store this many connections will be allocated when the traffic manager starts.

```java
void setL4AccelMaxConcurrentConnections(
    Unsigned Integer value
)
```
**setL4AccelMaxConcurrentConnectionsByLocation( location, value ) throws**

InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of concurrent connections, in millions, that can be handled by each L4Accel child process. An appropriate amount of memory to store this many connections will be allocated when the traffic manager starts. This is a location specific function, any action will operate on the specified location.

```java
void setL4AccelMaxConcurrentConnectionsByLocation(
    String location,
    Unsigned Integer value
)
```

**setLicenseServers( values ) throws InvalidInput, DeploymentError**

Set A list of license servers for FLA licensing.

```java
void setLicenseServers(
    String[] values
)
```

**setLicenseServersByLocation( location, values ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set A list of license servers for FLA licensing. This is a location specific function, any action will operate on the specified location.

```java
void setLicenseServersByLocation(
    String location,
    String[] values
)
```

**setListenQueueSize( value ) throws InvalidInput, DeploymentError**

Set the size of the listen queue for managing incoming connections.

```java
void setListenQueueSize(
    Unsigned Integer value
)
```

**setListenQueueSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the size of the listen queue for managing incoming connections. This is a location specific function, any action will operate on the specified location.

```java
void setListenQueueSizeByLocation(
    String location,
    Unsigned Integer value
)
```

**setLogExportAuthHTTP( value ) throws InvalidInput, DeploymentError**

Set the HTTP authentication method to use when exporting log entries.

```java
void setLogExportAuthHTTP(
    GlobalSettings.LogExportAuthHTTP value
)
```
setLogExportAuthHTTPByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the HTTP authentication method to use when exporting log entries. This is a location specific function, any action will operate on the specified location.

void setLogExportAuthHTTPByLocation(
    String location
    GlobalSettings.LogExportAuthHTTP value
)

setLogExportAuthHecToken( value ) throws InvalidInput, DeploymentError

Set the HTTP Event Collector token to use for HTTP authentication with a Splunk server.

void setLogExportAuthHecToken(
    String value
)

setLogExportAuthHecTokenByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the HTTP Event Collector token to use for HTTP authentication with a Splunk server. This is a location specific function, any action will operate on the specified location.

void setLogExportAuthHecTokenByLocation(
    String location
    String value
)

setLogExportAuthPassword( value ) throws InvalidInput, DeploymentError

Set the password to use for HTTP basic authentication.

void setLogExportAuthPassword(
    String value
)

setLogExportAuthPasswordByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the password to use for HTTP basic authentication. This is a location specific function, any action will operate on the specified location.

void setLogExportAuthPasswordByLocation(
    String location
    String value
)

setLogExportAuthUsername( value ) throws InvalidInput, DeploymentError

Set the username to use for HTTP basic authentication.

void setLogExportAuthUsername(
    String value
)
**setLogExportAuthUsernameByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the username to use for HTTP basic authentication. This is a location specific function, any action will operate on the specified location.

```java
void setLogExportAuthUsernameByLocation(
    String location
    String value
)
```

**setLogExportEnabled( value ) throws InvalidInput, DeploymentError**

Set whether to monitor log files and export entries to the configured endpoint.

```java
void setLogExportEnabled(
    Boolean value
)
```

**setLogExportEnabledByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set whether to monitor log files and export entries to the configured endpoint. This is a location specific function, any action will operate on the specified location.

```java
void setLogExportEnabledByLocation(
    String location
    Boolean value
)
```

**setLogExportEndpoint( value ) throws InvalidInput, DeploymentError**

Set the URL to which log entries should be sent.

```java
void setLogExportEndpoint(
    String value
)
```

**setLogExportEndpointByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the URL to which log entries should be sent. This is a location specific function, any action will operate on the specified location.

```java
void setLogExportEndpointByLocation(
    String location
    String value
)
```

**setLogExportRequestTimeout( value ) throws InvalidInput, DeploymentError**

Set the number of seconds after which HTTP requests sent to the configured endpoint will be considered to have failed if no response is received.

```java
void setLogExportRequestTimeout(
    Unsigned Integer value
)
```
setLogExportRequestTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of seconds after which HTTP requests sent to the configured endpoint will be considered to have failed if no response is received. This is a location specific function, any action will operate on the specified location.

void setLogExportRequestTimeoutByLocation(
    String location
    Unsigned Integer value
)

setLogExportTLSVerify( value ) throws InvalidInput, DeploymentError

Set whether the server certificate should be verified when connecting to the endpoint. If enabled, server certificates that do not match the server name, are self-signed, have expired, have been revoked, or that are signed by an unknown CA will be rejected.

void setLogExportTLSVerify(
    Boolean value
)

setLogExportTLSVerifyByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the server certificate should be verified when connecting to the endpoint. If enabled, server certificates that do not match the server name, are self-signed, have expired, have been revoked, or that are signed by an unknown CA will be rejected. This is a location specific function, any action will operate on the specified location.

void setLogExportTLSVerifyByLocation(
    String location
    Boolean 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
)

setLogFlushFlushTimeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the length of time to wait before flushing the request log files for each virtual server, in seconds. This is a location specific function, any action will operate on the specified location.

void setLogFlushFlushTimeByLocation(
    String location
    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
)
setLogIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the length of time between log messages for log intensive features e.g. SLM, in seconds. This is a location specific function, any action will operate on the specified location.

```java
void setLogIntervalByLocation(
    String location
    Unsigned Integer value
)
```

setLogRate( value ) throws InvalidInput, DeploymentError
Set is the maximum number of connection errors logged per second.

```java
void setLogRate(
    Unsigned Integer value
)
```

setLogRateByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set is the maximum number of connection errors logged per second. This is a location specific function, any action will operate on the specified location.

```java
void setLogRateByLocation(
    String location
    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.

```java
void setLogReopenTime(
    Unsigned Integer value
)
```

setLogReopenTimeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the length of time to wait before re-opening request log files, to handle log file rotation, in seconds. This is a location specific function, any action will operate on the specified location.

```java
void setLogReopenTimeByLocation(
    String location
    Unsigned Integer value
)
```

setLoginBanner( value ) throws InvalidInput, DeploymentError
Set the banner text to be shown on the Admin Server login page and before logging in to appliance SSH servers.

```java
void setLoginBanner(
    String value
)
```
**setLoginDelay( value ) throws InvalidInput, DeploymentError**

Set the number of seconds before another login attempt can be made after a failed attempt.

```java
void setLoginDelay(
    Unsigned Integer value
)
```

**setMaxAccepting( value ) throws InvalidInput, DeploymentError**

Set how many traffic manager child processes accept new connections.

```java
void setMaxAccepting(
    Unsigned Integer value
)
```

**setMaxAcceptingByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set how many traffic manager child processes accept new connections. This is a location specific function, any action will operate on the specified location.

```java
void setMaxAcceptingByLocation(
    String location
    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.

```java
void setMaxIdleConnections(
    Unsigned Integer value
)
```

**setMaxIdleConnectionsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the maximum number of unused HTTP keepalive connections to all nodes that should maintained for re-use. This is a location specific function, any action will operate on the specified location.

```java
void setMaxIdleConnectionsByLocation(
    String location
    Unsigned Integer value
)
```

**setMaxKeepalives( value ) throws InvalidInput, DeploymentError**

setMaxKeepalives is deprecated, please use setMaxIdleConnections instead.

```java
void setMaxKeepalives(
    Unsigned Integer value
)
```

**setMaxKeepalivesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

setMaxKeepalives is deprecated, please use setMaxIdleConnections instead. This is a location specific function, any action will operate on the specified location.
Function Reference

GlobalSettings

voidsetMaxKeepalivesByLocation{
    String location
    Unsigned Integer value
}

setMaxLoginAttempts( value ) throws InvalidInput, DeploymentError
Set the number of sequential failed login attempts that will cause a user account to be suspended. Setting this to 0 disables this feature.

voidsetMaxLoginAttempts{
    Unsigned Integer value
}

setMaxLoginExternal( value ) throws InvalidInput, DeploymentError
Set whether or not usernames blocked due to the max_login_attempts limit should also be blocked from authentication against external services (such as LDAP and RADIUS).

voidsetMaxLoginExternal{
    Boolean value
}

setMaxLoginSuspensionTime( value ) throws InvalidInput, DeploymentError
Set number of minutes to suspend users who have exceeded the max_login_attempts limit.

voidsetMaxLoginSuspensionTime{
    Unsigned Integer value
}

setMaxRetries( value ) throws InvalidInput, DeploymentError
This method is now obsolete and is replaced by Pool.setNodeConnectionAttempts.

voidsetMaxRetries{
    Unsigned Integer value
}

setMaximumFDCount( value ) throws InvalidInput, DeploymentError
Set the maximum number of file descriptors that your traffic manager will allocate.

voidsetMaximumFDCount{
    Unsigned Integer value
}

setMaximumFDCountByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum number of file descriptors that your traffic manager will allocate. This is a location specific function, any action will operate on the specified location.

voidsetMaximumFDCountByLocation{
    String location
    Unsigned Integer value
}
setMinAlphaChars( value ) throws InvalidInput, DeploymentError
Set the minimum number of alphabetic characters in a password.

void setMinAlphaChars(
    Unsigned Integer value
)

setMinNumericChars( value ) throws InvalidInput, DeploymentError
Set the minimum number of numeric characters in a password.

void setMinNumericChars(
    Unsigned Integer value
)

setMinPasswordLength( value ) throws InvalidInput, DeploymentError
Set the minimum number of characters a password must contain.

void setMinPasswordLength(
    Unsigned Integer value
)

setMinSpecialChars( value ) throws InvalidInput, DeploymentError
Set the minimum number of special characters in a password.

void setMinSpecialChars(
    Unsigned Integer value
)

setMinUppercaseChars( value ) throws InvalidInput, DeploymentError
Set the minimum number of uppercase characters in a password.

void setMinUppercaseChars(
    Unsigned Integer value
)

setMonitorNumNodes( value ) throws InvalidInput, DeploymentError
Set the maximum number of nodes, pools and locations that can be monitored.

void setMonitorNumNodes(
    Unsigned Integer value
)

setMonitorNumNodesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum number of nodes, pools and locations that can be monitored. This is a location specific function, any action will operate on the specified location.

void setMonitorNumNodesByLocation(
    String location
    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.

```java
void setMultipleAccept(
    Boolean value
)
```

**setMultipleAcceptByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set whether your traffic manager should try and read multiple new connections each time a new client connects. This is a location specific function, any action will operate on the specified location.

```java
void setMultipleAcceptByLocation(
    String location
    Boolean value
)
```

**setNodeConnectionAttempts( value ) throws InvalidInput, DeploymentError**

This method is now obsolete and is replaced by Pool.setNodeConnectionAttempts.

```java
void setNodeConnectionAttempts(
    Unsigned Integer value
)
```

**setNodeFailTime( value ) throws InvalidInput, DeploymentError**

This method is now obsolete and is replaced by Pool.setNodeFailTime.

```java
void setNodeFailTime(
    Unsigned Integer value
)
```

**setOCSPCacheSize( value ) throws InvalidInput, DeploymentError**

Set the maximum number of cached client certificate OCSP results stored. This cache is used to speed up OCSP checks against client certificates by caching results.

```java
void setOCSPCacheSize(
    Unsigned Integer value
)
```

**setOCSPCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the maximum number of cached client certificate OCSP results stored. This cache is used to speed up OCSP checks against client certificates by caching results. This is a location specific function, any action will operate on the specified location.

```java
void setOCSPCacheSizeByLocation(
    String location
    Unsigned Integer value
)
```

**setOspfv2Area( value ) throws InvalidInput, DeploymentError**

Set the OSPF area in which the traffic manager will operate.
void setOspfv2Area(String value)

setOspfv2AreaByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the OSPF area in which the traffic manager will operate. This is a location specific function, any action will operate on the specified location.

void setOspfv2AreaByLocation(String location, String value)

setOspfv2AreaType(value) throws InvalidInput, DeploymentError

Set the type of OSPF area

void setOspfv2AreaType(GlobalSettings.Ospfv2AreaType value)

setOspfv2AreaTypeByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the type of OSPF area This is a location specific function, any action will operate on the specified location.

void setOspfv2AreaTypeByLocation(String location, GlobalSettings.Ospfv2AreaType value)

setOspfv2AuthenticationKeyIdA(value) throws InvalidInput, DeploymentError

Set the OSPF key ID

void setOspfv2AuthenticationKeyIdA(Unsigned Integer value)

setOspfv2AuthenticationKeyIdAByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the OSPF key ID This is a location specific function, any action will operate on the specified location.

void setOspfv2AuthenticationKeyIdAByLocation(String location, Unsigned Integer value)

setOspfv2AuthenticationKeyIdB(value) throws InvalidInput, DeploymentError

Set the OSPF key ID

void setOspfv2AuthenticationKeyIdB(Unsigned Integer value)
**Function Reference**

**GlobalSettings**

**setOspfv2AuthenticationKeyIdBByLocation( location, value )** throws **InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the OSPF key ID. This is a location specific function, any action will operate on the specified location.

```java
void setOspfv2AuthenticationKeyIdBByLocation(
    String location
    Unsigned Integer value
)
```

**setOspfv2AuthenticationSharedSecretA( value )** throws **InvalidInput, DeploymentError**

Set the OSPF MD5 shared secret, set to "" to disable.

```java
void setOspfv2AuthenticationSharedSecretA(
    String value
)
```

**setOspfv2AuthenticationSharedSecretAByLocation( location, value )** throws **InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the OSPF MD5 shared secret, set to "" to disable. This is a location specific function, any action will operate on the specified location.

```java
void setOspfv2AuthenticationSharedSecretAByLocation(
    String location
    String value
)
```

**setOspfv2AuthenticationSharedSecretB( value )** throws **InvalidInput, DeploymentError**

Set the OSPF MD5 shared secret, set to "" to disable.

```java
void setOspfv2AuthenticationSharedSecretB(
    String value
)
```

**setOspfv2AuthenticationSharedSecretBBByLocation( location, value )** throws **InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the OSPF MD5 shared secret, set to "" to disable. This is a location specific function, any action will operate on the specified location.

```java
void setOspfv2AuthenticationSharedSecretBBByLocation(
    String location
    String value
)
```

**setOspfv2Enabled( value )** throws **InvalidInput, DeploymentError**

Set whether OSPF Route Health Injection is enabled

```java
void setOspfv2Enabled(
    Boolean value
)
```
setOspfv2EnabledByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether OSPF Route Health Injection is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setOspfv2EnabledByLocation(
    String location
    Boolean value
)
```

setOspfv2HelloInterval( value ) throws InvalidInput, DeploymentError

Set the interval at which OSPF "hello" packets are sent to the network.

```java
void setOspfv2HelloInterval(
    Unsigned Integer value
)
```

setOspfv2HelloIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the interval at which OSPF "hello" packets are sent to the network. This is a location specific function, any action will operate on the specified location.

```java
void setOspfv2HelloIntervalByLocation(
    String location
    Unsigned Integer value
)
```

setOspfv2RouterDeadInterval( value ) throws InvalidInput, DeploymentError

Set the number of seconds before declaring a silent router down.

```java
void setOspfv2RouterDeadInterval(
    Unsigned Integer value
)
```

setOspfv2RouterDeadIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of seconds before declaring a silent router down. This is a location specific function, any action will operate on the specified location.

```java
void setOspfv2RouterDeadIntervalByLocation(
    String location
    Unsigned Integer value
)
```

setPasswordChangesPerDay( value ) throws InvalidInput, DeploymentError

Set the maximum number of times a password can be changed every 24 hours.

```java
void setPasswordChangesPerDay(
    Unsigned Integer value
)
```

setPasswordReuseAfter( value ) throws InvalidInput, DeploymentError

Set the number of times a password must have been changed before it can be reused.
void setPasswordReuseAfter(
    Unsigned Integer value
)

setPostLoginBanner( value ) throws InvalidInput, DeploymentError
Set the banner text to be displayed on the appliance console after login.

void setPostLoginBanner(
    String value
)

setProtectionConncountSize( value ) throws InvalidInput, DeploymentError
Set the amount of shared memory reserved for an inter-process table of combined connection counts used by Service Protection classes (specified as an absolute size, eg 20MB).

void setProtectionConncountSize(
    String value
)

setProtectionConncountSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the amount of shared memory reserved for an inter-process table of combined connection counts used by Service Protection classes (specified as an absolute size, eg 20MB). This is a location specific function, any action will operate on the specified location.

void setProtectionConncountSizeByLocation(
    String location
    String value
)

setRESTAuthTimeout( value ) throws InvalidInput, DeploymentError
Set REST authentication timeout.

void setRESTAuthTimeout(
    Unsigned Integer value
)

setRESTEnabled( value ) throws InvalidInput, DeploymentError
Set whether REST service is enabled.

void setRESTEnabled(
    Boolean value
)

setRESTMaxHTTPHeaderLength( value ) throws InvalidInput, DeploymentError
Set the maximum allowed length in bytes of a HTTP request's headers.

void setRESTMaxHTTPHeaderLength(
    Unsigned Integer value
)

setRESTReplicateAbsoluteTime( value ) throws InvalidInput, DeploymentError
Set Absolute time before configuration replication via REST.
void setRESTReplicateAbsoluteTime(
    Unsigned Integer value
)

setRESTReplicateLullTime( value ) throws InvalidInput, DeploymentError
Set Lull time for configuration replication via REST.
void setRESTReplicateLullTime(
    Unsigned Integer value
)

setRESTReplicateTimeout( value ) throws InvalidInput, DeploymentError
Set the configuration replication timeout via REST.
void setRESTReplicateTimeout(
    Unsigned Integer value
)

setRateClassLimit( value ) throws InvalidInput, DeploymentError
Set the maximum number of Rate classes allowed.
void setRateClassLimit(
    Unsigned Integer value
)

setRateClassLimitByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum number of Rate classes allowed. This is a location specific function, any action will
operate on the specified location.
void setRateClassLimitByLocation(
    String location
    Unsigned Integer value
)

setRecentConns( value ) throws InvalidInput, DeploymentError
Set the details of how many recently closed connections each traffic manager process should save for use
with the Connections page.
void setRecentConns(
    Unsigned Integer value
)

setRecentConnsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the details of how many recently closed connections each traffic manager process should save for use
with the Connections page. This is a location specific function, any action will operate on the specified
location.
void setRecentConnsByLocation(
    String location
    Unsigned Integer value
)
setRecentConnsRetainTime( value ) throws InvalidInput, DeploymentError

Set for how long a snapshot should be retained on the Connections page.

void setRecentConnsRetainTime(
    Unsigned Integer value
)

setRecentConnsRetainTimeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set for how long a snapshot should be retained on the Connections page. This is a location specific function, any action will operate on the specified location.

void setRecentConnsRetainTimeByLocation(
    String location
    Unsigned Integer value
)

setRecentConnsSnapshotSize( value ) throws InvalidInput, DeploymentError

Set the maximum number of connections each traffic manager process should show for a snapshot on the Connections page.

void setRecentConnsSnapshotSize{
    Unsigned Integer value
}

setRecentConnsSnapshotSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of connections each traffic manager process should show for a snapshot on the Connections page. This is a location specific function, any action will operate on the specified location.

void setRecentConnsSnapshotSizeByLocation{
    String location
    Unsigned Integer value
}

setSLMClassLimit( value ) throws InvalidInput, DeploymentError

Set the maximum number of SLM classes allowed.

void setSLMClassLimit{
    Unsigned Integer value
}

setSLMClassLimitByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of SLM classes allowed. This is a location specific function, any action will operate on the specified location.

void setSLMClassLimitByLocation{
    String location
    Unsigned Integer value
}
setSNATIPLimit( value ) throws InvalidInput, DeploymentError

Set the maximum number of Source NAT IP addresses that can be used across all Traffic IP Groups.

```java
void setSNATIPLimit(
    Unsigned Integer value
)
```

setSNATIPLimitByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of Source NAT IP addresses that can be used across all Traffic IP Groups. This is a location specific function, any action will operate on the specified location.

```java
void setSNATIPLimitByLocation(
    String location,
    Unsigned Integer value
)
```

setSNATIPLocalPortRangeHigh( value ) throws InvalidInput, DeploymentError

Set the upper boundary of the port range reserved for use by the kernel. Ports above this range will be used by the traffic manager for establishing outgoing connections.

```java
void setSNATIPLocalPortRangeHigh(
    Unsigned Integer value
)
```

setSNATIPLocalPortRangeHighByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the upper boundary of the port range reserved for use by the kernel. Ports above this range will be used by the traffic manager for establishing outgoing connections. This is a location specific function, any action will operate on the specified location.

```java
void setSNATIPLocalPortRangeHighByLocation(
    String location,
    Unsigned Integer value
)
```

setSNATSharedPoolSize( value ) throws InvalidInput, DeploymentError

Set the size of the Source NAT shared memory pool used for shared storage across child processes.

```java
void setSNATSharedPoolSize(
    Unsigned Integer value
)
```

setSNATSharedPoolSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the size of the Source NAT shared memory pool used for shared storage across child processes. This is a location specific function, any action will operate on the specified location.

```java
void setSNATSharedPoolSizeByLocation(
    String location,
    Unsigned Integer value
)
```
**setSNMPUserCounters( value ) throws InvalidInput, DeploymentError**

Set the number of user defined SNMP counters (this single parameter dictates the numbers of both 32- and 64-bit user counters - there is always the same number of counters of each type).

```java
void setSNMPUserCounters(
    Unsigned Integer value
)
```

**setSNMPUserCountersByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the number of user defined SNMP counters (this single parameter dictates the numbers of both 32- and 64-bit user counters - there is always the same number of counters of each type). This is a location specific function, any action will operate on the specified location.

```java
void setSNMPUserCountersByLocation(
    String location
    Unsigned Integer value
)
```

**setSSL3AllowRehandshake( value ) throws InvalidInput, DeploymentError**

Set whether SSL / TLS re-handshakes are supported.

```java
void setSSL3AllowRehandshake(
    GlobalSettings.SSL3AllowRehandshake value
)
```

**setSSL3AllowRehandshakeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set whether SSL / TLS re-handshakes are supported. This is a location specific function, any action will operate on the specified location.

```java
void setSSL3AllowRehandshakeByLocation(
    String location
    GlobalSettings.SSL3AllowRehandshake 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).

```java
void setSSL3Ciphers(
    String value
)
```

**setSSL3CiphersByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the list of configured SSL ciphers (available ciphers can be displayed using the command $ZEUSHOME/zxtm/bin/zeus.zxtm -s). This is a location specific function, any action will operate on the specified location.

```java
void setSSL3CiphersByLocation(
    String location
    String value
)```
setSSL3DiffieHellmanKeyLength( value ) throws InvalidInput, DeploymentError

Set the number of bits to use for Diffie-Hellman keys.

```java
void setSSL3DiffieHellmanKeyLength(
    GlobalSettings.SSL3DiffieHellmanKeyLength value
)
```

setSSL3DiffieHellmanKeyLengthByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of bits to use for Diffie-Hellman keys. This is a location specific function, any action will operate on the specified location.

```java
void setSSL3DiffieHellmanKeyLengthByLocation(
    String location
    GlobalSettings.SSL3DiffieHellmanKeyLength value
)
```

setSSL3MinRehandshakeInterval( value ) throws InvalidInput, DeploymentError

Set the minimum time interval (in milliseconds) between handshakes on a single SSL3/TLS connection.

```java
void setSSL3MinRehandshakeInterval(
    Unsigned Integer value
)
```

setSSLAzureClientID( value ) throws InvalidInput, DeploymentError

Set the client identifier for the Azure Key Vault.

```java
void setSSLAzureClientID(
    String value
)
```

setSSLAzureClientIDByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the client identifier for the Azure Key Vault. This is a location specific function, any action will operate on the specified location.

```java
void setSSLAzureClientIDByLocation(
    String location
    String value
)
```

setSSLAzureClientSecret( value ) throws InvalidInput, DeploymentError

Set the client secret for the Azure Key Vault.

```java
void setSSLAzureClientSecret(
    String value
)
```

setSSLAzureClientSecretByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the client secret for the Azure Key Vault. This is a location specific function, any action will operate on the specified location.
Function Reference

GlobalSettings

void setSSLAzureClientSecretByLocation(
    String location
    String value
)

setSSLAzureVaultURL( value ) throws InvalidInput, DeploymentError
Set the URL of the Azure Key Vault REST API.
void setSSLAzureVaultURL{
    String value
}

setSSLAzureVaultURLByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the URL of the Azure Key Vault REST API. This is a location specific function, any action will operate on the specified location.
void setSSLAzureVaultURLByLocation{
    String location
    String value
}

setSSLAzureVerifyRESTAPICert( value ) throws InvalidInput, DeploymentError
Set whether the SSL certificate of the Azure Key Vault REST API will be verified using CAs present in the SSL Certificate Authorities Catalog.
void setSSLAzureVerifyRESTAPICert{
    Boolean value
}

setSSLAzureVerifyRESTAPICertByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set whether the SSL certificate of the Azure Key Vault REST API will be verified using CAs present in the SSL Certificate Authorities Catalog. This is a location specific function, any action will operate on the specified location.
void setSSLAzureVerifyRESTAPICertByLocation{
    String location
    Boolean value
}

setSSLCRLMemSize( value ) throws InvalidInput, DeploymentError
Set the size of the CRL shared memory.
void setSSLCRLMemSize{
    String value
}

setSSLCRLMemSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the size of the CRL shared memory. This is a location specific function, any action will operate on the specified location.
void setSSLCRLMemSizeByLocation{
GlobalSettings Function Reference

**String location**

**String value**

)  

**setSSLDFailureCount( value )** throws InvalidInput, DeploymentError  

setSSLDFailureCount is deprecated, please use setSSLHardwareFailureCount instead.

```java
void setSSLDFailureCount(
    Unsigned Integer value
)
```

**setSSLDFailureCountByLocation( location, value )** throws InvalidInput, DeploymentError, ObjectDoesNotExist  

setSSLDFailureCount is deprecated, please use setSSLHardwareFailureCount instead. This is a location specific function, any action will operate on the specified location.

```java
void setSSLDFailureCountByLocation(
    String location
    Unsigned Integer value
)
```

**setSSLPKCS11Lib( value )** throws InvalidInput, DeploymentError  

setSSLPKCS11Lib is deprecated, please use setSSLHardwarePKCS11Lib instead.

```java
void setSSLPKCS11Lib(
    String value
)
```

**setSSLPKCS11LibByLocation( location, value )** throws InvalidInput, DeploymentError, ObjectDoesNotExist  

setSSLPKCS11Lib is deprecated, please use setSSLHardwarePKCS11Lib instead. This is a location specific function, any action will operate on the specified location.

```java
void setSSLPKCS11LibByLocation(
    String location
    String value
)
```

**setSSLPKCS11UserPIN( value )** throws InvalidInput, DeploymentError  

setSSLPKCS11UserPIN is deprecated, please use setSSLHardwarePKCS11UserPIN instead.

```java
void setSSLPKCS11UserPIN(
    String value
)
```

**setSSLPKCS11UserPINByLocation( location, value )** throws InvalidInput, DeploymentError, ObjectDoesNotExist  

setSSLPKCS11UserPIN is deprecated, please use setSSLHardwarePKCS11UserPIN instead. This is a location specific function, any action will operate on the specified location.

```java
void setSSLPKCS11UserPINByLocation(
    String location
    String value
)
```
setSSLEllipticCurves( value ) throws InvalidInput, DeploymentError
Set the elliptic curve preference list for SSL connections unless overridden by virtual server or pool settings

void setSSLEllipticCurves(
    String value
)

setSSLEllipticCurvesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the elliptic curve preference list for SSL connections unless overridden by virtual server or pool settings. This is a location specific function, any action will operate on the specified location.

void setSSLEllipticCurvesByLocation(
    String location
    String value
)

setSSLHardwareAccelerator( value ) throws InvalidInput, DeploymentError
Set whether your traffic manager should always attempt to use SSL hardware.

void setSSLHardwareAccelerator(
    Boolean value
)

setSSLHardwareAcceleratorByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set whether your traffic manager should always attempt to use SSL hardware. This is a location specific function, any action will operate on the specified location.

void setSSLHardwareAcceleratorByLocation(
    String location
    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
)

setSSLHardwareFailureCountByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the number of consecutive failures from the SSL hardware that will be tolerated before your traffic manager tries to log in again. This is a location specific function, any action will operate on the specified location.

void setSSLHardwareFailureCountByLocation(
    String location
    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
)

setSSLHardwarePKCS11LibByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the location of the PKCS#11 library supplied by your hardware vendor. This is a location specific function, any action will operate on the specified location.

void setSSLHardwarePKCS11LibByLocation(
    String location
    String value
)

setSSLHardwarePKCS11SlotLabel( value ) throws InvalidInput, DeploymentError
Set the label of the SSL hardware slot to use.

void setSSLHardwarePKCS11SlotLabel(
    String value
)

setSSLHardwarePKCS11SlotLabelByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the label of the SSL hardware slot to use. This is a location specific function, any action will operate on the specified location.

void setSSLHardwarePKCS11SlotLabelByLocation(
    String location
    String value
)

setSSLHardwarePKCS11SlotType( value ) throws InvalidInput, DeploymentError
Set the type of PKCS11 slot to use. Only used for PKCS11.

void setSSLHardwarePKCS11SlotType(
    GlobalSettings.SSLHardwarePKCS11SlotType value
)

setSSLHardwarePKCS11SlotTypeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the type of PKCS11 slot to use. Only used for PKCS11. This is a location specific function, any action will operate on the specified location.

void setSSLHardwarePKCS11SlotTypeByLocation(
    String location
    GlobalSettings.SSLHardwarePKCS11SlotType value
)

setSSLHardwarePKCS11UserPIN( value ) throws InvalidInput, DeploymentError
Set the user PIN for the PKCS token (PKCS#11 devices only)
void setSSLHardwarePKCS11UserPIN(String value)

setSSLHardwarePKCS11UserPINByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the user PIN for the PKCS token (PKCS#11 devices only). This is a location-specific function, any action will operate on the specified location.

void setSSLHardwarePKCS11UserPINByLocation(String location, String value)

setSSLHardwareType(value) throws InvalidInput, DeploymentError

Set the device driver library name.

void setSSLHardwareType(String value)

setSSLHardwareTypeByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the device driver library name. This is a location-specific function, any action will operate on the specified location.

void setSSLHardwareTypeByLocation(String location, String value)

setSSLHonorFallbackSCSV(value) throws InvalidInput, DeploymentError

Set whether ssl-decrypting Virtual Servers honor the Fallback SCSV

void setSSLHonorFallbackSCSV(Boolean value)

setSSLHonorFallbackSCSVByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether ssl-decrypting Virtual Servers honor the Fallback SCSV. This is a location-specific function, any action will operate on the specified location.

void setSSLHonorFallbackSCSVByLocation(String location, Boolean value)

setSSLInsertExtraFragment(value) throws InvalidInput, DeploymentError

Set whether SSL3 and TLS1 use one byte fragments

void setSSLInsertExtraFragment(Boolean value)
setSSLInsertExtraFragmentByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether SSL3 and TLS1 use one byte fragments. This is a location specific function, any action will operate on the specified location.

```java
void setSSLInsertExtraFragmentByLocation(
    String location
    Boolean value
)
```

setSSLMaxHandshakeMessageSize(value) throws InvalidInput, DeploymentError

Set the maximum acceptable size (in bytes) a SSL handshake message is permitted to be.

```java
void setSSLMaxHandshakeMessageSize(
    Unsigned Integer value
)
```

setSSLMaxHandshakeMessageSizeByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum acceptable size (in bytes) a SSL handshake message is permitted to be. This is a location specific function, any action will operate on the specified location.

```java
void setSSLMaxHandshakeMessageSizeByLocation(
    String location
    Unsigned Integer value
)
```

setSSLOCSPStaplingDefaultRefreshInterval(value) throws InvalidInput, DeploymentError

Set how long to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling, if we don’t have an up-to-date OCSP response.

```java
void setSSLOCSPStaplingDefaultRefreshInterval(
    Unsigned Integer value
)
```

setSSLOCSPStaplingDefaultRefreshIntervalByLocation(location, value) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how long to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling, if we don’t have an up-to-date OCSP response. This is a location specific function, any action will operate on the specified location.

```java
void setSSLOCSPStaplingDefaultRefreshIntervalByLocation(
    String location
    Unsigned Integer value
)
```

setSSLOCSPStaplingMaximumRefreshInterval(value) throws InvalidInput, DeploymentError

Set maximum number of seconds to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling. (0 means no maximum.)

```java
void setSSLOCSPStaplingMaximumRefreshInterval(
```
setSSLOCSPStaplingMaximumRefreshIntervalByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set maximum number of seconds to wait before refreshing requests on behalf of the store of certificate status responses used by OCSP stapling. (0 means no maximum.) This is a location specific function, any action will operate on the specified location.

void setSSLOCSPStaplingMaximumRefreshIntervalByLocation{
    String location
    Unsigned Integer value
}

setSSLOCSPStaplingMemSize( value ) throws InvalidInput, DeploymentError

Set the size of the OCSP stapling response shared memory.

void setSSLOCSPStaplingMemSize{
    String value
}

setSSLOCSPStaplingMemSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the size of the OCSP stapling response shared memory. This is a location specific function, any action will operate on the specified location.

void setSSLOCSPStaplingMemSizeByLocation{
    String location
    String value
}

setSSLOCSPStaplingTimeTolerance( value ) throws InvalidInput, DeploymentError

Set how many seconds to allow the current time to be outside the validity time of an OCSP response before considering it invalid.

void setSSLOCSPStaplingTimeTolerance{
    Unsigned Integer value
}

setSSLOCSPStaplingTimeToleranceByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how many seconds to allow the current time to be outside the validity time of an OCSP response before considering it invalid. This is a location specific function, any action will operate on the specified location.

void setSSLOCSPStaplingTimeToleranceByLocation{
    String location
    Unsigned Integer value
}

setSSLOCSPStaplingVerifyResponse( value ) throws InvalidInput, DeploymentError

Set whether to verify the OCSP response signature before caching a response for OCSP stapling.
setSSLOCSPStaplingVerifyResponse {  
  Boolean value
}

setSSLOCSPStaplingVerifyResponseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether to verify the OCSP response signature before caching a response for OCSP stapling. This is a location specific function, any action will operate on the specified location.

setSSLPreventTimingSideChannels( value ) throws InvalidInput, DeploymentError

Set whether SSL3 and TLS will take performance degrading steps to prevent exposing timing side-channels.

setSSLPreventTimingSideChannelsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether SSL3 and TLS will take performance degrading steps to prevent exposing timing side-channels. This is a location specific function, any action will operate on the specified location.

setSSLSessionCachePerVirtualserver( value ) throws InvalidInput, DeploymentError

Set whether an SSL session created by a given virtual server can only be resumed by a connection to the same virtual server.

setSSLSessionCachePerVirtualserverByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether an SSL session created by a given virtual server can only be resumed by a connection to the same virtual server. This is a location specific function, any action will operate on the specified location.

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.
Function Reference

GlobalSettings

void setSSLSessionCacheSize(
    Unsigned Integer value
)

setSSLSessionCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of entries in the SSL session cache. This is used to provide persistence based on SSL session IDs. This is a location specific function, any action will operate on the specified location.

void setSSLSessionCacheSizeByLocation(
    String location
    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
)

setSSLSessionIDCacheExpiryTimeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the length of time that SSL session IDs are stored, in seconds. This is a location specific function, any action will operate on the specified location.

void setSSLSessionIDCacheExpiryTimeByLocation(
    String location
    Unsigned Integer value
)

setSSLSessionIDCacheSize( value ) throws InvalidInput, DeploymentError

Set the number of entries in the SSL session ID cache.

void setSSLSessionIDCacheSize(
    Unsigned Integer value
)

setSSLSessionIDCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of entries in the SSL session ID cache. This is a location specific function, any action will operate on the specified location.

void setSSLSessionIDCacheSizeByLocation(
    String location
    Unsigned Integer value
)

setSSLSignatureAlgorithms( value ) throws InvalidInput, DeploymentError

Set the SSL signature algorithms preference list for SSL connections unless overridden by virtual server or pool settings.

void setSSLSignatureAlgorithms(
    String value
)
GlobalSettings Function Reference

setSSLSignatureAlgorithmsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the SSL signature algorithms preference list for SSL connections unless overridden by virtual server or pool settings. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSignatureAlgorithmsByLocation(
    String location
    String value
)
```

setSSLSupportSSL2( value ) throws InvalidInput, DeploymentError

This method is now deprecated.

```java
void setSSLSupportSSL2(
    Boolean value
)
```

setSSLSupportSSL2ByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

This method is now deprecated. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportSSL2ByLocation(
    String location
    Boolean value
)
```

setSSLSupportSSL3( value ) throws InvalidInput, DeploymentError

Set whether SSLv3 support is enabled.

```java
void setSSLSupportSSL3(
    Boolean value
)
```

setSSLSupportSSL3ByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether SSLv3 support is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportSSL3ByLocation(
    String location
    Boolean value
)
```

setSSLSupportTLS1( value ) throws InvalidInput, DeploymentError

Set whether TLSv1 support is enabled.

```java
void setSSLSupportTLS1(
    Boolean value
)
```
setSSLSupportTLS11( value ) throws InvalidInput, DeploymentError

Set whether TLSv1.1 support is enabled.

```java
void setSSLSupportTLS11(
    Boolean value
)
```

setSSLSupportTLS11ByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether TLSv1.1 support is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportTLS11ByLocation(
    String location
    Boolean value
)
```

setSSLSupportTLS12( value ) throws InvalidInput, DeploymentError

Set whether TLSv1.2 support is enabled.

```java
void setSSLSupportTLS12(
    Boolean value
)
```

setSSLSupportTLS12ByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether TLSv1.2 support is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportTLS12ByLocation(
    String location
    Boolean value
)
```

setSSLSupportTLS1ByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether TLSv1 support is enabled. This is a location specific function, any action will operate on the specified location.

```java
void setSSLSupportTLS1ByLocation(
    String location
    Boolean value
)
```

setSharedPoolSize( value ) throws InvalidInput, DeploymentError

Set is the size of shared memory pool to be used for shared storage across worker processes.

```java
void setSharedPoolSize(
    String value
)
```
setSharedPoolSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set is the size of shared memory pool to be used for shared storage across worker processes. This is a location specific function, any action will operate on the specified location.

```java
void setSharedPoolSizeByLocation(
    String location
    String value
)
```

setSoapIdleMinutes( value ) throws InvalidInput, DeploymentError

Set the number of minutes the SOAP server remain idle before exiting

```java
void setSoapIdleMinutes(
    Unsigned Integer value
)
```

setSoapIdleMinutesByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of minutes the SOAP server remain idle before exiting This is a location specific function, any action will operate on the specified location.

```java
void setSoapIdleMinutesByLocation(
    String location
    Unsigned Integer value
)
```

setSocketOptimizations( value ) throws InvalidInput, DeploymentError

Set whether potential network socket optimisations should be used.

```java
void setSocketOptimizations(
    GlobalSettings.SocketOptimizations value
)
```

setSocketOptimizationsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether potential network socket optimisations should be used. This is a location specific function, any action will operate on the specified location.

```java
void setSocketOptimizationsByLocation(
    String location
    GlobalSettings.SocketOptimizations value
)
```

setSsldAccel( value ) throws InvalidInput, DeploymentError

setSsldAccel is deprecated, please use setSSLHardwareAccelerator instead.

```java
void setSsldAccel(
    Boolean value
)
```
**Function Reference**

**GlobalSettings**

```java
setSsldAccelByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

setSsldAccel is deprecated, please use setSSLHardwareAccelerator instead. This is a location specific function, any action will operate on the specified location.

void setSsldAccelByLocation(
    String location
    Boolean value
)

setSsldLibrary( value ) throws InvalidInput, DeploymentError

setSsldLibrary is deprecated, please use setSSLHardwareType instead.

void setSsldLibrary(
    GlobalSettings.SsldLibrary value
)

setSsldLibraryByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

setSsldLibrary is deprecated, please use setSSLHardwareType instead. This is a location specific function, any action will operate on the specified location.

void setSsldLibraryByLocation(
    String location
    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
)

setStateSyncTimeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set how often the cache state is propagated to other traffic managers in the cluster, in seconds. This is a location specific function, any action will operate on the specified location.

void setStateSyncTimeByLocation(
    String location
    Unsigned Integer value
)

setStateSyncTimeout( value ) throws InvalidInput, DeploymentError

Set the timeout for state propagation between cluster members, in seconds

void setStateSyncTimeout(
    Unsigned Integer value
)
setStateSyncTimeoutByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the timeout for state propagation between cluster members, in seconds. This is a location specific function, any action will operate on the specified location.

```java
void setStateSyncTimeoutByLocation(
    String location
    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).

```java
void setSystemReadBufferSize(
    Unsigned Integer value
)
```

setSystemReadBufferSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the size of the operating system’s read buffer, in bytes (0 means use the system default). This is a location specific function, any action will operate on the specified location.

```java
void setSystemReadBufferSizeByLocation(
    String location
    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).

```java
void setSystemWriteBufferSize(
    Unsigned Integer value
)
```

setSystemWriteBufferSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the size of the operating system’s write buffer, in bytes (0 means use the system default). This is a location specific function, any action will operate on the specified location.

```java
void setSystemWriteBufferSizeByLocation(
    String location
    Unsigned Integer value
)
```

setTrackUnknownUsers( value ) throws InvalidInput, DeploymentError

Set whether to remember past login attempts from usernames that are not known to exist (should be No for an Admin Server accessible from the public Internet).

```java
void setTrackUnknownUsers(
    Boolean value
)
```
setTrafficIPGroupLimit( value ) throws InvalidInput, DeploymentError

Set the maximum number of Traffic IP Groups allowed.

```java
void setTrafficIPGroupLimit(
    Unsigned Integer value
)
```

setTrafficIPGroupLimitByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the maximum number of Traffic IP Groups allowed. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficIPGroupLimitByLocation(
    String location
    Unsigned Integer value
)
```

setTrafficScriptExecutionTimeWarning( value ) throws InvalidInput, DeploymentError

Set the number of milliseconds a rule can run for before a warning is logged.

```java
void setTrafficScriptExecutionTimeWarning(
    Unsigned Integer value
)
```

setTrafficScriptExecutionTimeWarningByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the number of milliseconds a rule can run for before a warning is logged. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficScriptExecutionTimeWarningByLocation(
    String location
    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.

```java
void setTrafficScriptMemoryWarning(
    Unsigned Integer value
)
```

setTrafficScriptMemoryWarningByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the amount of buffered network data a TrafficScript rule can buffer before a warning is logged, in bytes. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficScriptMemoryWarningByLocation(
    String location
    Unsigned Integer value
)
```
**setTrafficscriptArrayElements( value ) throws InvalidInput, DeploymentError**

Set the number of array elements that can be stored before additional memory is allocated.

```java
void setTrafficscriptArrayElements{
    Unsigned Integer value
}
```

**setTrafficscriptArrayElementsByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the number of array elements that can be stored before additional memory is allocated. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficscriptArrayElementsByLocation{
    String location
    Unsigned Integer value
}
```

**setTrafficscriptDataLocalSize( value ) throws InvalidInput, DeploymentError**

Set the maximum size of the TrafficScript local data pool (specified as a percentage of system RAM, e.g. ‘5%’, or an absolute size, e.g. 200MB)

```java
void setTrafficscriptDataLocalSize{
    String value
}
```

**setTrafficscriptDataLocalSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the maximum size of the TrafficScript local data pool (specified as a percentage of system RAM, e.g. ‘5%’, or an absolute size, e.g. 200MB) This is a location specific function, any action will operate on the specified location.

```java
void setTrafficscriptDataLocalSizeByLocation{
    String location
    String 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)

```java
void setTrafficscriptDataSize{
    String value
}
```

**setTrafficscriptDataSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

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) This is a location specific function, any action will operate on the specified location.

```java
void setTrafficscriptDataSizeByLocation{
    String location
    String value
}
Function Reference

GlobalSettings

**setTrafficscriptMaxInstr( value ) throws InvalidInput, DeploymentError**

Set the maximum number of instructions a TrafficScript rule will run before being aborted.

```java
void setTrafficscriptMaxInstr(
    Unsigned Integer value
)
```

**setTrafficscriptMaxInstrByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the maximum number of instructions a TrafficScript rule will run before being aborted. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficscriptMaxInstrByLocation(
    String location
    Unsigned Integer value
)
```

**setTrafficscriptRegexCacheSize( value ) throws InvalidInput, DeploymentError**

Set the number of regular expressions to cache

```java
void setTrafficscriptRegexCacheSize(
    Unsigned Integer value
)
```

**setTrafficscriptRegexCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the number of regular expressions to cache. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficscriptRegexCacheSizeByLocation(
    String location
    Unsigned Integer value
)
```

**setTrafficscriptRegexMatchLimit( value ) throws InvalidInput, DeploymentError**

Set the maximum number of ways TrafficScript will attempt to match a regular expression at each position in the subject string, before it aborts the rule and reports a TrafficScript error.

```java
void setTrafficscriptRegexMatchLimit(
    Unsigned Integer value
)
```

**setTrafficscriptRegexMatchLimitByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist**

Set the maximum number of ways TrafficScript will attempt to match a regular expression at each position in the subject string, before it aborts the rule and reports a TrafficScript error. This is a location specific function, any action will operate on the specified location.

```java
void setTrafficscriptRegexMatchLimitByLocation(
    String location
    Unsigned Integer value
)
setTrafficscriptRegexMatchWarnPerc( value ) throws InvalidInput, DeploymentError

Set the percentage of trafficscript!regex_match_limit at which TrafficScript reports a performance warning.

void setTrafficscriptRegexMatchWarnPerc(
    Unsigned Integer value
)

setTrafficscriptRegexMatchWarnPercByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the percentage of trafficscript!regex_match_limit at which TrafficScript reports a performance warning. This is a location specific function, any action will operate on the specified location.

void setTrafficscriptRegexMatchWarnPercByLocation(
    String location
    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
)

setTrafficscriptVariablePoolUseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the 'pool.use' and 'pool.select' TrafficScript functions accept variables as well as literal strings. This is a location specific function, any action will operate on the specified location.

void setTrafficscriptVariablePoolUseByLocation(
    String location
    Boolean value
)

setTransactionExportEnabled( value ) throws InvalidInput, DeploymentError

Set whether to export metadata about transactions processed by the traffic manager to an external location.

void setTransactionExportEnabled(
    Boolean value
)

setTransactionExportEnabledByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether to export metadata about transactions processed by the traffic manager to an external location. This is a location specific function, any action will operate on the specified location.

void setTransactionExportEnabledByLocation(
    String location
    Boolean value
)
setTransactionExportEndpoint( value ) throws InvalidInput, DeploymentError

Set the endpoint to which transaction metadata should be exported.

```java
void setTransactionExportEndpoint(
    String value
)
```

setTransactionExportEndpointByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set the endpoint to which transaction metadata should be exported. This is a location specific function, any action will operate on the specified location.

```java
void setTransactionExportEndpointByLocation(
    String location
    String value
)
```

setTransactionExportTLS( value ) throws InvalidInput, DeploymentError

Set whether the connection to the specified endpoint should be encrypted.

```java
void setTransactionExportTLS(
    Boolean value
)
```

setTransactionExportTLSByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the connection to the specified endpoint should be encrypted. This is a location specific function, any action will operate on the specified location.

```java
void setTransactionExportTLSByLocation(
    String location
    Boolean value
)
```

setTransactionExportTLSVerify( value ) throws InvalidInput, DeploymentError

Set whether the server certificate presented by the endpoint should be verified, preventing a connection from being established if the certificate does not match the server name, is self-signed, is expired, is revoked, or has an unknown CA.

```java
void setTransactionExportTLSVerify(
    Boolean value
)
```

setTransactionExportTLSVerifyByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether the server certificate presented by the endpoint should be verified, preventing a connection from being established if the certificate does not match the server name, is self-signed, is expired, is revoked, or has an unknown CA. This is a location specific function, any action will operate on the specified location.

```java
void setTransactionExportTLSVerifyByLocation(
    String location
    Boolean value
)
```
setUipageBanner( value ) throws InvalidInput, DeploymentError
Set the banner text to be displayed on all Admin Server pages.

void setUipageBanner(
    String value
)

setUniversalSessionCacheSize( value ) throws InvalidInput, DeploymentError
Set the maximum number of entries in the universal session cache.

void setUniversalSessionCacheSize(
    Unsigned Integer value
)

setUniversalSessionCacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum number of entries in the universal session cache. This is a location specific function, any action will operate on the specified location.

void setUniversalSessionCacheSizeByLocation(
    String location
    Unsigned Integer value
)

setWebcacheAvgPathLength( value ) throws InvalidInput, DeploymentError
Set the estimated average length of the path for resources to be cached.

void setWebcacheAvgPathLength(
    Unsigned Integer value
)

setWebcacheAvgPathLengthByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the estimated average length of the path for resources to be cached. This is a location specific function, any action will operate on the specified location.

void setWebcacheAvgPathLengthByLocation(
    String location
    Unsigned Integer value
)

setWebcacheDisk( value ) throws InvalidInput, DeploymentError
Set whether the webcache is stored on disk.

void setWebcacheDisk(
    Boolean value
)

setWebcacheDiskByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set whether the webcache is stored on disk. This is a location specific function, any action will operate on the specified location.
void setWebcacheDiskByLocation{
   String location
   Boolean value
}

setWebcacheDiskDir( value ) throws InvalidInput, DeploymentError
Set the disk cache location
void setWebcacheDiskDir{
   String value
}

setWebcacheDiskDirByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the disk cache location This is a location specific function, any action will operate on the specified location.
void setWebcacheDiskDirByLocation{
   String location
   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
}

setWebcacheMaxFileNumByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum number of files that can be stored in the web cache This is a location specific function, any action will operate on the specified location.
void setWebcacheMaxFileNumByLocation{
   String location
   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
}

setWebcacheMaxFileSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
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'. This is a location specific function, any action will operate on the specified location.
void setWebcacheMaxFileSizeByLocation{
string location
  string value
)

setWebcacheMaxPathLength( value ) throws InvalidInput, DeploymentError
Set the maximum length of the path for the resource being cached.

void setWebcacheMaxPathLength(
  unsigned integer value
)

setWebcacheMaxPathLengthByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set the maximum length of the path for the resource being cached. This is a location specific function, any action will operate on the specified location.

void setWebcacheMaxPathLengthByLocation(
  string location
  unsigned integer 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
)

setWebcacheNormalizeQueryByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
Set whether the assignment sub-strings in the parameter string are put into alphabetical order. This is a location specific function, any action will operate on the specified location.

void setWebcacheNormalizeQueryByLocation(
  string location
  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
)

setWebcacheSizeByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist
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) This is a location specific function, any action will operate on the specified location.

void setWebcacheSizeByLocation(
  string location
String value
)

setWebcacheVerbose( value ) throws InvalidInput, DeploymentError

Set whether an X-Cache-Info header to show cacheability should be added.

void setWebcacheVerbose(
    Boolean value
)

setWebcacheVerboseByLocation( location, value ) throws InvalidInput, DeploymentError, ObjectDoesNotExist

Set whether an X-Cache-Info header to show cacheability should be added. This is a location specific function, any action will operate on the specified location.

void setWebcacheVerboseByLocation(
    String location
    Boolean value
)

Structures

GlobalSettings.ReturnPathRoute

This structure contains a return path route, consisting of MAC + IPv4 + IPv6 addresses. At least one IP address is required.

struct GlobalSettings.ReturnPathRoute  {
    # The MAC address of the router.
    String mac;

    # The IPv4 address of the router.
    String ipv4;

    # The IPv6 address of the router.
    String ipv6;
}

Enumerations

GlobalSettings.AdminAllowRehandshake

enum GlobalSettings.AdminAllowRehandshake {
    # Always allow
    always,

    # Allow safe re-handshakes
    safe,

    # Only if client uses RFC 5746 (Secure Renegotiation Extension)
    rfc5746,

    # Never allow
    never
}
GlobalSettings

GlobalSettings.AdminDiffieHellmanKeyLength

enum GlobalSettings.AdminDiffieHellmanKeyLength {
    # 1024
dh_1024,

    # 2048
dh_2048,

    # 3072
dh_3072,

    # 4096
dh_4096
}

GlobalSettings.DataPlaneAccelerationCores

enum GlobalSettings.DataPlaneAccelerationCores {
    # 1
One,

    # 2
Two,

    # 4
Four
}

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.LogExportAuthHTTP

enum GlobalSettings.LogExportAuthHTTP {
    # None
none,

    # Basic (Username and Password)
Function Reference

GlobalSettings

    basic,
    # Splunk {HBC token}
    splunk

}  

**GlobalSettings.Ospfv2AreaType**

enum GlobalSettings.Ospfv2AreaType {
    # Normal area
    normal,
    # Stub area
    stub,
    # Not So Stubby Area (RFC3101)
    nssa
}

**GlobalSettings.SSL3AllowRehandshake**

denum GlobalSettings.SSL3AllowRehandshake {
    # Always allow
    always,
    # Allow safe re-handshakes
    safe,
    # Only if client uses RFC 5746 (Secure Renegotiation Extension)
    rfc5746,
    # Never allow
    never
}

**GlobalSettings.SSL3DiffieHellmanKeyLength**

denum GlobalSettings.SSL3DiffieHellmanKeyLength {
    # 1024
    dh_1024,
    # 2048
    dh_2048,
    # 3072
    dh_3072,
    # 4096
    dh_4096
}

**GlobalSettings.SSLHardwarePKCS11SlotType**

denum GlobalSettings.SSLHardwarePKCS11SlotType {
    # Operator Card Set
    operator,
    # Soft Card
    softcard,
    # Module Protected
    module
GlobalSettings.SSLHardwareType

enum GlobalSettings.SSLHardwareType {
    # None
    none,
    # PKCS#11 (e.g. nCipher NetHSM)
    pkcs11,
    # Microsoft Azure Key Vault
    azure
}

GlobalSettings.SocketOptimizations

enum GlobalSettings.SocketOptimizations {
    # auto
    auto,
    # Yes
    Yes,
    # No
    No
}

GlobalSettings.SsldLibrary

enum GlobalSettings.SsldLibrary {
    # None
    none,
    # PKCS#11 (e.g. nCipher NetHSM)
    pkcs11,
    # Microsoft Azure Key Vault
    azure
}

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.

Methods

deleteFile( names ) throws ObjectDoesNotExist

Delete the named files from the conf/extra directory.
void deleteFile(
downloadFile( name ) throws ObjectDoesNotExist

Download the named file from the conf/extra directory.

```java
Binary Data downloadFile(
    String name
)
```

getFileNames()

Get the names of all the files stored in the conf/extra directory.

```java
String[] getFileNames()
```

uploadFile( name, content ) throws InvalidObjectName

Uploads a new file into the conf/extra directory, overwriting the file if it already exists.

```java
void uploadFile(
    String name
    Binary Data content
)
```

---

**Diagnose**

URI: http://soap.zeus.com/zxtm/1.1/Diagnose/

The Diagnose interface provides information about errors and problems in the system.

**Methods**

activateTrafficManagers( hostnames ) throws InvalidInput

Activate traffic managers that have recovered from failures and are ready to start taking Traffic IPs.

```java
void activateTrafficManagers(
    String[] hostnames
)
```

diagnoseSystem()

Provides all diagnostic information about the system.

```java
Diagnose.ErrorInfo diagnoseSystem()
```

getInactiveTrafficManagers()

List the traffic managers that have recovered from failures and are ready to start taking Traffic IPs.

```java
String[] getInactiveTrafficManagers()
```
**getTechnicalSupportReport()**

Download a technical support report suitable for providing to your support provider to help diagnose problems.

Binary Data `getTechnicalSupportReport()`

**Structures**

**Diagnose.AgeError**

This structure combines an error message with its age in seconds.

```c
struct Diagnose.AgeError  {
    # Seconds since the error occurred
    Integer age;
    # error message
    String error;
}
```

**Diagnose.ConfigError**

This structure contains information about configuration errors.

```c
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).

```c
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 node failures.

```java
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.

```java
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.

```java
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;
}
```
Enumerations

Diagnose.ErrLevel

This enumeration defines the possible severity levels of an error.

```java
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
}
```

System.Backups

URI: http://soap.zeus.com/zxtm/1.0/System/Backups/

The Backups interfaces provide operations on saved configuration backup archives.

Methods

**createBackup(name, description ) throws ObjectAlreadyExists, InvalidObjectName**

Create backup archive based on the current configuration

```java
void createBackup(
    String name
    String description
)
```

**deleteAllBackups()**

Delete all the backups

```java
void deleteAllBackups()
```

**deleteBackups(names ) throws ObjectDoesNotExist**

Delete one or more backups

```java
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
)
```

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;
}
```
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.

Methods

addCloudcredentialNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Cloud Credentials that will trigger the specified event types. If the event type has no Cloud Credentials names configured, all objects of this type will match.

```java
void addCloudcredentialNames(
    String[] names
    String[][] objects
)
```

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.

```java
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.

```java
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.

```java
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.

```java
void addLicensekeyNames(
    String[] names
    String[][][] objects
)
```

addLocationNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of Location that will trigger the specified event types. If the event type has no Location names configured, all objects of this type will match.

```java
void addLocationNames(
    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.

```java
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.

```java
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.

```java
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.
Alerting.EventType Function Reference

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
)

addServiceNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Add the names of GLB Service that will trigger the specified event types. If the event type has no GLB Service names configured, all objects of this type will match.

void addServiceNames(
    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.

```java
void addZxtmNames(
    String[] names,
    String[][] objects
)
```

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

Copy each of the named event types.

```java
void copyEventType(
    String[] names,
    String[] new_names
)
```

deleteEventType( names ) throws ObjectDoesNotExist, DeploymentError

Removes one or more event types.

```java
void deleteEventType(
    String[] names
)
```

getCloudcredentialNames( names ) throws ObjectDoesNotExist

Get the names of Cloud Credentials that will trigger the specified event types. If the event type has no Cloud Credentials names configured, all objects of this type will match.

```java
String[][] getCloudcredentialNames(
    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.

```java
String[][] getCustomEvents(
    String[] names
)
```

gETEventType( names ) throws ObjectDoesNotExist

Returns a set of event type objects for the specified names.

```java
Alerting.EventType.EventType[] getEventType(
    String[] names
)
```

gETEventTypeNames()

Returns the names of all event types in the system.

```java
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.

```java
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.

```java
String[][] getLicensekeyNames(
    String[] names
)
```

getLocationNames( names ) throws ObjectDoesNotExist

Get the names of Location that will trigger the specified event types. If the event type has no Location names configured, all objects of this type will match.

```java
String[][] getLocationNames(
    String[] names
)
```

getMappedActions( names ) throws ObjectDoesNotExist

Get an action that will be run when this event type is triggered.

```java
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.

```java
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.

```java
String[][] getNodeNames(
    String[] names
)
```

getNote( names ) throws ObjectDoesNotExist

Get the note for each of the named Event Types.

```java
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
)
```

ggetServiceNames( names ) throws ObjectDoesNotExist

Get the names of GLB Service that will trigger the specified event types. If the event type has no GLB Service names configured, all objects of this type will match.

```
String[][] getServiceNames(
    String[] names
)
```

ggetSlmNames( 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
)
```

ggetVserverNames( 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
)
```

ggetZxtmNames( 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
)
Alerting.EventType Function Reference

removeCloudcredentialNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Cloud Credentials that will trigger the specified event types. If the event type has no Cloud Credentials names configured, all objects of this type will match.

```java
void removeCloudcredentialNames(
    String[] names
    String[][] objects
)
```

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.

```java
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.

```java
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.

```java
void removeLicensekeyNames(
    String[] names
    String[][] objects
)
```

removeLocationNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of Location that will trigger the specified event types. If the event type has no Location names configured, all objects of this type will match.

```java
void removeLocationNames(
    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.

```java
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.

```java
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.

```java
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.

```java
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.

```java
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
)

removeServiceNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Remove the names of GLB Service that will trigger the specified event types. If the event type has no GLB Service names configured, all objects of this type will match.

void removeServiceNames(
    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
)
**setCloudCredentialNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError**

Set the names of Cloud Credentials that will trigger the specified event types. If the event type has no Cloud Credentials names configured, all objects of this type will match.

```java
void setCloudCredentialNames(
    String[] names
    String[][] objects
)
```

**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.

```java
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.

```java
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.

```java
void setLicensekeyNames(
    String[] names
    String[][] objects
)
```

**setLocationNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError**

Set the names of Location that will trigger the specified event types. If the event type has no Location names configured, all objects of this type will match.

```java
void setLocationNames(
    String[] names
    String[][] objects
)
```
setMappedActions(String[] names, String[][] values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set an action that will be run when this event type is triggered.

```java
void setMappedActions(
    String[] names
    String[][] values
)
```

setMonitorNames(String[] names, String[][] 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.

```java
void setMonitorNames(
    String[] names
    String[][] objects
)
```

setNodeNames(String[] names, String[][] 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.

```java
void setNodeNames(
    String[] names
    String[][] events
)
```

setNote(String[] names, String[] values) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the note for each of the named Event Types.

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

setPoolNames(String[] names, String[][] 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.

```java
void setPoolNames(
    String[] names
    String[][] objects
)
```

setProtectionNames(String[] names, String[][] 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.

```java
void setProtectionNames(
    String[] names
    String[][] objects
)
```
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
)

setServiceNames( names, objects ) throws InvalidInput, ObjectDoesNotExist, InvalidOperation, DeploymentError

Set the names of GLB Service that will trigger the specified event types. If the event type has no GLB Service names configured, all objects of this type will match.

void setServiceNames(
    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
)
Alerting.EventType

A set of conditions that when met causes an action to be run.

```java
struct Alerting.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 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 Cloud Credentials that should trigger this event type. If this is an empty array all objects of this type will be matched.
    String[] cloudcredentialNames;

    # 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;

    # The names of all the Locations that should trigger this event type. If this is an empty array all objects of this type will be matched.
    String[] locationNames;

    # 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 GLB Services that should trigger this event type. If this is an empty array all objects of this type will be matched.
    String[] serviceNames;

    # 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;
}
```
Enumerations

Alerting.EventType

enum Alerting.EventType.Event {
    # This event matches all events.
    ALL,

    # Special value that matches all events of type Cloud Credentials.
    cloudcredentials_ALL,

    # Cloud Credentials - A cloud API process querying changes to cloud instances
    # is hanging
    cloudcredentials_apistatusprocesshanging,

    # Cloud Credentials - An API call made by the autoscaler process has returned
    # a response that could not be parsed
    cloudcredentials_autoscalerresponseparseerror,

    # Cloud Credentials - An API call made by the autoscaler process has reported
    # an error
    cloudcredentials_autoscalestatusupdateerror,

    # Cloud Credentials - A cloud API process has timed out
    cloudcredentials_autoscalingprocessstimedout,

    # Cloud Credentials - A Cloud Credentials object has been deleted but it was
    # still in use
    cloudcredentials_usedcredsdeleted,

    # 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
    faulttoleranceActivatedAutomatically,

    # Fault Tolerance - All machines are working
    faulttolerance_allmachinesok,

    # Fault Tolerance - Automatic failback after delay
    faulttolerance_autofailbackafterdelay,

    # Fault Tolerance - Auto-failback delay timer cancelled
Alerting.EventType Function Reference

- faulttolerance_autofailbacktimercancelled,
- faulttolerance_autofailbacktimerstarted,
- faulttolerance_autofailbacktimerstopped,
- faulttolerance_bgpneighborsdegraded,
- faulttolerance_bgpneighborsfailed,
- faulttolerance_bgpneighborsok,
- faulttolerance_bgpnoneighbors,
- faulttolerance_clockjump,
- faulttolerance_clocknotmonotonic,
- faulttolerance_dropec2ipwarn,
- faulttolerance_dropipinfo,
- faulttolerance_dropipwarn,
- faulttolerance_ec2flipperraiselocalworking,
- faulttolerance_ec2flipperraiseothersdead,
- faulttolerance_ec2iperr,
- faulttolerance_ec2nopublicip,
- faulttolerance_ec2nosecondaryprivateip,
- faulttolerance_flipperbackendsworking,
- faulttolerance_flipperdadreraise,
- faulttolerance_flipperdadraise,
- faulttolerance_frontendsworking.
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 - This Traffic Manager has re-raised traffic IP addresses
# as the remote machine which was hosting them has dropped them
faulttolerance_flipperraiseremotedropped,

# 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 - Some of the monitored OSPF neighbors are not peered
faulttolerance_ospfneighborldegraded,

# Fault Tolerance - None of the monitored OSPF neighbors are peered
faulttolerance_ospfneighborsfailed,

# Fault Tolerance - All monitored OSPF neighbors are peered
faulttolerance_ospfneighborsok,

# 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 - Failed to send ping packets
faulttolerance_pingsendfail,

# Fault Tolerance - Routing software had a major failure and will be
# restarted
Alerting.EventType Function Reference

faulttolerance_routingswfailed,
# Fault Tolerance - Routing software has failed and reached its failure limit
faulttolerance_routingswfailurelimitreached,

faulttolerance_routingswoperational,
# Fault Tolerance - Routing software is now operational

faulttolerance_routingswstartfailed,
# Fault Tolerance - Routing software failed to start

faulttolerance_statebaddata,
# Fault Tolerance - Received an invalid response from another cluster member

faulttolerance_stateconnfail,
# Fault Tolerance - Failed to connect to another cluster member for state
# sharing

faulttolerance_statereadfail,
# Fault Tolerance - Reading state data from another cluster member failed

faulttolerance_statetimeout,
# Fault Tolerance - Timeout while sending state data to another cluster
# member

faulttolerance_stateunexpected,
# Fault Tolerance - Writing state data to another cluster member failed

faulttolerance_statereadfail,
# Fault Tolerance - An error occurred when using the zcluster Multi-Hosted IP
# kernel module

general_ALL,
# Special value that matches all events of type General.

generalAppStateerror,
# General - Application firewall control command failed

general_appfirewallcontrolrestarted,
# General - Application firewall restarted

general_appfirewallcontrolstarted,
# General - Application firewall started

general_appfirewallcontrolstoppered,
# General - Application firewall control command timed out

general_appfirewallcontroltimeout,
# General - Appliance notification

general_appliance,
# General - An audit log event has occurred

genrer_audit,
Function Reference

Alerting.EventType

# General - An error occurred during user authentication
general_autherror,

# General - A hostname used for DNS-derived Autoscaling doesn't resolve
general_autoscaleresolvefailure,

# General - Autoscaling not permitted by licence key
general_autoscalinglicenseerror,

# General - There was an error communicating with a child process
general_childcommsfail,

# General - Replication of configuration has failed
general_confrepfailed,

# General - Replication of configuration has timed out
general_confreptimeout,

# General - The built-in DNS server has failed to create a DNS record
general_dnszonecreaterecord,

# General - The built-in DNS server has failed to parse a DNS zone file
general_dnszoneparse,

# General - The built-in DNS server has failed to validate a DNS zone file
general_dnszonevalidate,

# General - Data Plane Acceleration mode diagnostics
general_dpadiagnostics,

# General - Data Plane Acceleration mode error
general_dpaerror,

# General - Traffic manager failed to get the required data from Amazon servers
general_ec2dataretrievalfailed,

# General - Traffic manager has now successfully retrieved the required data from Amazon servers
general_ec2dataretrievalsuccessful,

# General - The EC2 instance is now initialized
general_ec2initialized,

# General - Running out of free file descriptors
general_fewfreefds,

# General - FIPS 140-2 cryptographic module initialization failed
general_fipsfailinit,

# General - FIPS 140-2 cryptographic module operations failed
general_fipsfailops,

# General - Traffic manager failed to get the required data from GCE instance
general_gcedataretrievalfailed,

# General - Traffic manager has now successfully retrieved the required data from GCE instance
general_gcedataretrievalsuccessful,

# General - Failed to load geolocation data
general_geodataloadfail,

# General - A location has been disabled because you have exceeded the
# licence limit
general_licensesetoomanylocations,

# General - Log disk partition full
general_logdiskfull,

# General - Log disk partition usage has exceeded threshold
general_logdiskoverload,

# General - DNS-derived Autoscaling will resume updating, as the DNS server
# is now responding
general_nameserveravailable,

# General - DNS-derived Autoscaling will not update, as the DNS server is
# unavailable
general_nameserverunavailable,

# General - Total number of locations exceeded the maximum limit
general_numlocations-exceeded,

# General - Total number of nodes exceeded the maximum number of nodes that
# can be monitored
general_numnodes-exceeded,

# General - Total number of pools exceeded the maximum limit
general_numpools-exceeded,

# General - Total number of traffic IP group exceeded the maximum limit
general_numtipg-exceeded,

# General - OCSP request (for OCSP stapling) failed
general_ocspstaplingfail,

# General - Insufficient memory for OCSP stapling
general_ocspstaplingnomem,

# General - An OCSP request (for OCSP stapling) reported that a certificate
# was revoked
general_ocspstaplingrevoked,

# General - An OCSP request (for OCSP stapling) reported that a certificate
# was unknown
general_ocspstaplingunknown,

# General - An old but good OCSP response was returned for a revoked
# certificate
general_ocspstaplingunrevoked,

# General - Software must be restarted to apply configuration changes
general_restartrequired,

# General - Software is running
general_running,

# General - CRL does not fit in the configured amount of shared memory,
# increase ssl!crl_mem!size and restart software
general_sslcrltoobig,

# General - Virtual Traffic Manager Appliance reboot required
general_sysctlreboot,

# General - Time has been moved back
general_timemovedback,
# General - Virtual Traffic Manager restart/reboot required
general_unspecifiedreboot,

# General - Virtual Traffic Manager Appliance reboot required
general_upgradereboot,

# General - Virtual Traffic Manager software restart required
general_upgraderestart,

# General - The number of simultaneously active connections has reached a
# level that the software cannot process in due time because of CPU
# starvation; there is a high risk of connections timing out
general_zxtmcpustarvation,

# General - The number of simultaneously active connections has reached a
# level that the software cannot process in due time; there is a high risk of
# connections timing out
general_zxtmhighload,

# General - Internal software error
general_zxtmswerror,

# Special value that matches all events of type Java.
java_ALL,

# Java - Java runner died
java_javadied,

# Java - Cannot start Java runner, program not found
java_javanotfound,

# Java - Java runner started
java_javastarted,

# Java - Java runner failed to start
java_javastartfail,

# Java - Java support has stopped
java.javastop,

# Java - Java runner 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 - Analytics Export support has been disabled
licensekeys_analyticsexportlicensedisabled,

# License Keys - Analytics Export support has been enabled
licensekeys_analyticsexportlicenseenabled,

# License Keys - Realtime Analytics support has been disabled
licensekeys_analyticslicensingdisabled,

# License Keys - Realtime Analytics support has been enabled
licensekeys_analyticslicenseenabled,

# License Keys - Autoscaling support has been disabled
licensekeys_autoscalinglicensedisabled,
Alerting.EventType Function Reference

# License Keys - Autoscaling support has been enabled
licensekeys AutoscalingLicenseEnabled,

# 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 7 days
licensekeys_expiresoon,

# License Keys - License key expires within 15 days
licensekeys_expiresoon15,

# License Keys - License key expires within 30 days
licensekeys_expiresoon30,

# License Keys - License key expires within 60 days
licensekeys_expiresoon60,

# License Keys - License key expires within 90 days
licensekeys_expiresoon90,

# License Keys - License allows less memory for caching
licensekeys_lessmemallowed,

# License Keys - License key authorized
licensekeys_license-authorized,

# License Keys - License key authorized by authorization code
licensekeys_license-authorized-ts,

# License Keys - License key explicitly disabled from authorization code
licensekeys_license-explicitlydisabled-ts,

# License Keys - Unable to authorize license key
licensekeys_license-graceperiodexpired,

# License Keys - Unable to authorize license key
licensekeys_license-graceperiodexpired-ts,

# License Keys - License server rejected license key; key remains authorized
licensekeys_license-rejected-authorized,

# License Keys - License key rejected from authorization code; key remains
# authorized
licensekeys_license-rejected-authorized-ts,

# License Keys - License server rejected license key; key is not authorized
licensekeys_license-rejected-unauthorized,

# License Keys - License key rejected from authorization code
licensekeys_license-rejected-unauthorized-ts,

# License Keys - Unable to contact license server; license key remains
# authorized
licensekeys_license-timedout-authorized,

# License Keys - Unable to run authorization code to completion; key remains
# valid
licensekeys_license-timedout-authorized-ts,

# License Keys - Unable to contact license server; license key is not authorized
licensekeys_license-timedout-authorized,

# License Keys - Unable to run authorization code to completion
licensekeys_license-timedout-unauthorized-ts,

# License Keys - License key is not authorized
licensekeys_license-authorized,

# License Keys - Cluster size exceeds license key limit
licensekeys_licenseclustertoo-big,

# License Keys - License key is corrupt
licensekeys_licensecorrupt,

# License Keys - Error detected in LicenseStateFile format
licensekeys_licensestate-malformed,

# License Keys - Unable to preserve license state
licensekeys_licensestate-write-failed,

# License Keys - License allows more memory for caching
licensekeys_morenemallowed,

# 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 Locations.
locations_ALL,

# Locations - Location is now available for GLB Service
locations_locationavailable,

# Locations - Location has been disabled for GLB Service
locations_locationdisabled,

# Locations - Location is being drained for GLB Service
locations_locationdraining,

# Locations - Location has just been enabled for GLB Service
locations_locationenabled,

# Locations - Location has failed for GLB Service
locations_locationfail,

# Locations - A monitor has detected a failure in this location
locations_locationmonitorfail,
# Alerting.EventType Function Reference

## # Locations - A monitor has indicated this location is now working
locations_locationmonitorok,

## # Locations - Location is not being drained for GLB Service
locations_locationnotdraining,

## # Locations - Location is now healthy for GLB Service
locations_locationok,

## # Locations - An external SOAP agent has detected a failure in this location
locations_locationsoapfail,

## # Locations - An external SOAP agent indicates this location is now working
locations_locationsoapok,

## # Locations - Location has become unavailable for GLB Service
locations_locationunavailable,

## # Locations - Location no longer contains any machines
locations_locempty,

## # Locations - Machine now in location
locations_locmovemachine,

## # 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 - API change process still running after refractory period is over
pools_apichangeprocesshanging,

## # Pools - The creation of a new node requested by an autoscaled pool is now complete
pools_autonodecreationcomplete,

## # Pools - Creation of new node instigated
pools_autonodecreationstarted,

## # Pools - A cloud API call to destroy a node has been started
pools_autonodedestroyed,

## # Pools - The destruction of a node in an autoscaled pool is now complete
pools_autonodedestructioncomplete,

## # Pools - A node in an autoscaled pool has disappeared from the cloud
pools_autonodedisappeared,

## # Pools - IP address of newly created instance already existed in pool's node
pools_autonodeexisted,

## # Pools - Node has no public IP address
pools_autonodenopublicip,

## # Pools - A node in an DNS-derived autoscaled pool has been removed
pools_autonoderemoved,
# Pools - The status of a node in an autoscaled pool has changed
pools_autonodestatuschange,

# Pools - Two pools are trying to use the same instance
pools_autoscalednodecontested,

# Pools - An autoscaled pool is now refractory
pools_autoscaledpoolrefractory,

# Pools - Over maximum size - shrinking
pools_autoscaledpooltoobig,

# Pools - Minimum size undercut - growing
pools_autoscaledpooltoosmall,

# Pools - The 'imageid' was empty when attempting to create a node in an
# autoscaled pool
pools_autoscaleinvalidargforcreatenode,

# Pools - ‘unique id’ was empty when attempting to destroy a node in an
# autoscaled pool
pools_autoscaleinvalidargfordeletenode,

# Pools - A pool config file has been updated by the autoscaler process
pools_autoscalepoolconfupdate,

# Pools - A node created by the autoscaler has the wrong imageid
pools_autoscalewrongimageid,

# Pools - A node created by the autoscaler has a non-matching name
pools_autoscalewrongname,

# Pools - A node created by the autoscaler has the wrong sizeid
pools_autoscalewrongsizeid,

# Pools - An API process that should have created or destroyed a node has
# failed to produce the expected result
pools_autoscalingchangeprocessfailure,

# Pools - Autoscaling for a pool has been disabled due to errors
# communicating with the cloud API
pools_autoscalingdisabled,

# Pools - Minimum size reached, cannot shrink further
pools_autoscalinghitfloor,

# Pools - Maximum size reached by autoscaled pool, cannot grow further
pools_autoscalinghitroof,

# Pools - An autoscaled pool is waiting to grow
pools_autoscalinghysteresiscantgrow,

# Pools - An autoscaled pool is waiting to shrink
pools_autoscalinghysteresiscantshrink,

# Pools - An autoscaled pool's state has changed
pools_autoscalingpoolstatechange,

# Pools - An autoscaled pool has failed completely
pools_autoscalingresuscitatepool,

# Pools - HTTP response contained an invalid Content-Length header
pools_badcontentlen,
Alerting.EventType Function Reference

# Pools - Attempt to scale down a pool that only had pending nodes or none at all
pools_cannotshrinkemptypool,

# Pools - Node returned invalid EHLO response
pools_ehloinvalid,

# Pools - Removed node is in use and will be drained
pools_nodedrainingtodelete,

# Pools - Draining to delete period for node has expired
pools_nodedrainingtodeletetimeout,

# 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 - Rule attempted to use Web Accelerator but it is not enabled
rules_optimizedisabled,

# Rules - Rule selected an unknown Web Accelerator profile
rules_optimizeuseunknownprofile,

# Rules - Rule selected an unknown Web Accelerator scope
rules_optimizeuseunknownscope,

# Rules - data.local.set() has run out of space
rules_datapoolstorefull,

# 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 - Rule exceeded execution time warning threshold
rules_ruleoverrun,

# 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,
Alerting EventType Function Reference

# Rules - Rule supplied too much data in HTTP stream
rules_rulestreamerrorTooMUCH,

# Rules - Rule encountered an XML error
rules_rulexmlerr,

# Rules - GLB service rule aborted during execution
rules_serviceruleabort,

# Rules - GLB service rule specified a location that has either failed or
# been marked as draining in the service configuration
rules_servicerulelocdead,

# Rules - GLB service rule specified a location that is not configured for
# the service
rules_servicerulelocnotconfigured,

# Rules - GLB service rule specified an unknown location
rules_servicerulelocunknown,

# Special value that matches all events of type GLB Services.
services_ALL,

# GLB Services - Active datacentre mismatches among cluster members
services_glbactivedcmismatch,

# GLB Services - A DNS Query returned IP addresses that are not configured
# for any location that is currently alive
services_glbdeadlocmissingips,

# GLB Services - Failed to alter DNS packet for global load balancing
services_glbfailalter,

# GLB Services - Failed to write log file for GLB service
services_glblogwritefail,

# GLB Services - Manual failback triggered
services_glbmanualfailback,

# GLB Services - A DNS Query returned IP addresses that are not configured in
# any location
services_glbmissingips,

# GLB Services - A location has been set as active for a GLB service
services_glbnewmaster,

# GLB Services - No valid location could be chosen for Global Load Balancing
services_glbnolocations,

# GLB Services - GLB Service has failed
services_glb servicedied,

# GLB Services - GLB Service has recovered
services_glb serviceok,

# GLB Services - There are too many Data Centers configured and the Global
# Load Balancing feature is not guaranteed to work reliably with more than
# 255 Data Centres
services_glb too many locations,

# Special value that matches all events of type SLM Classes.
slm_ALL,

# SLM Classes - SLM shared memory 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 - The built-in DNS server has successfully added a DNS zone
vservers_dnsaddzone,

# Virtual Servers - The built-in DNS server has failed to add a DNS zone
vservers_dnserroraddzone,

# Virtual Servers - The built-in DNS server has failed to delete a DNS zone
vservers_dnserrordeletezone,

# Virtual Servers - DNSSEC zone contains expired signatures
vservers_dnssecexpired,

# Virtual Servers - DNSSEC zone contains signatures that are about to expire
vservers_dnssecexpires,

# Virtual Servers - DNS zone has been deleted
vservers_dnszonedelete,

# Virtual Servers - A virtual server request log file was deleted (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_poolpersistence mismatch,

# Virtual Servers - Private key now OK (hardware available)
vservers_privkeyok,

# Virtual Servers - Error compressing HTTP response
vservers_responsetoolarge,

# Virtual Servers - Response headers from webserver too large
vservers_responsetoolarge,

# Virtual Servers - No suitable ports available for streaming data connection
vservers_sipstreamnoports,

# Virtual Servers - No suitable ports available for streaming data connection
vservers_hdstreamnoports,

# Virtual Servers - Request(s) received while SSL configuration invalid,
# connection closed
vservers_sslhandshakemsgsizelimit,

# Virtual Servers - SSL re-handshake requests have exceeded the frequency
# permitted by configuration
vservers_sslhandshakemsgsizelimit,

# Virtual Servers - Certificate Authority certificate expired
vservers_vscacertexpired,

# Virtual Servers - Certificate Authority certificate will expire within
# seven days
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 within seven days
vservers_vssslcerttoexpire,

# Virtual Servers - Virtual server started
vservers_vsstart,

# Virtual Servers - Virtual server stopped
vservers_vsstopt,

# Special value that matches all events of type Traffic Managers.
zxtms_ALL,
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.

Methods

addAction(names, types) throws InvalidInput, ObjectAlreadyExists, InvalidObjectName, DeploymentError

Add a action that can be run by an event.

```java
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'.

```java
void addScriptArguments(
    String[] names
    Alerting.Action.Argument[][] arguments
)
```

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

Copy each of the named actions.

```java
void copyAction(
    String[] names
    String[] new_names
)
```

deleteAction(names) throws ObjectDoesNotExist, ObjectInUse, DeploymentError

Deletes each of the named actions.

```java
void deleteAction(
    String[] names
)
```
**deleteActionProgram( names ) throws ObjectDoesNotExist, DeploymentError, ObjectInUse**

Delete the named action programs.

```java
void deleteActionProgram(
    String[] names
)
```

**downloadActionProgram( name ) throws ObjectDoesNotExist**

Download the named action program.

```java
Binary Data downloadActionProgram(
    String name
)
```

**getActionNames()**

Get the names of all available actions.

```java
String[] getActionNames()
```

**getActionNamesOfType( type )**

Get the names of all actions of the specified type.

```java
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.

```java
String[] getActionProgramNames()
```

**getActionType( names ) throws InvalidOperation, ObjectDoesNotExist**

Returns the type of each of the named actions.

```java
Alerting.Action.Type[] getActionType(
    String[] names
)
```

**getEmailRecipients( names ) throws InvalidOperation, ObjectDoesNotExist**

Get the address the alert emails are sent from.

```java
String[] getEmailRecipients(
    String[] names
)
```

**getEmailRecipientsByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist**

Get the address the alert emails are sent from. This is a location specific function, any action will operate on the specified location.

```java
String[] getEmailRecipientsByLocation(
    String location,
    String[] names
)
String location
String[] names
)

getEmailSMTPServer( names ) throws InvalidOperation, ObjectDoesNotExist
Get the SMTP server used to send alert emails for the specified actions.
String[] getEmailSMTPServer{
  String[] names
}

ggetEmailSMTPServerByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist
Get the SMTP server used to send alert emails for the specified actions. This is a location specific function, any action will operate on the specified location.
String[] getEmailSMTPServerByLocation{
  String location
  String[] names
}

ggetEmailSender( names ) throws InvalidOperation, ObjectDoesNotExist
Get the specified email addresses to the recipient list for the specified actions.
String[] getEmailSender{
  String[] names
}

ggetEmailSenderByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist
Get the specified email addresses to the recipient list for the specified actions. This is a location specific function, any action will operate on the specified location.
String[] getEmailSenderByLocation{
  String location
  String[] names
}

g.getLogFilePath( names ) throws InvalidOperation, ObjectDoesNotExist
Get the file this action logs to.
String[] getLogFilePath{
  String[] names
}

g.getLogFilePathByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist
Get the file this action logs to. This is a location specific function, any action will operate on the specified location.
String[] getLogFilePathByLocation{
  String location
  String[] names
}
Alerting.Action

getSNMPHashAlg( names ) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP hash algorithm for sending the Notify over SNMPv3. Valid values are "md5" and "sha1". The actions specified must be of type 'trap'.

Alerting.Action.SNMPHashAlg[] getSNMPHashAlg(
   String[] names
)

getSNMPHashAlgByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP hash algorithm for sending the Notify over SNMPv3. Valid values are "md5" and "sha1". The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

Alerting.Action.SNMPHashAlg[] getSNMPHashAlgByLocation(
   String location
   String[] names
)

getSNMPUsername( names ) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP username for sending the Notify over SNMPv3. The actions specified must be of type 'trap'.

String[] getSNMPUsername(
   String[] names
)

getSNMPUsernameByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP username for sending the Notify over SNMPv3. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

String[] getSNMPUsernameByLocation(
   String location
   String[] names
)

getSNMPVersion( names ) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP version used to send the trap/notify. The actions specified must be of type 'trap'.

Alerting.Action.SNMPVersion[] getSNMPVersion(
   String[] names
)

getSNMPVersionByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get the SNMP version used to send the trap/notify. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

Alerting.Action.SNMPVersion[] getSNMPVersionByLocation(
   String location
   String[] names
)
getSOAPAdditional( names ) throws InvalidOperation, ObjectDoesNotExist

Get the additional information to send with the SOAP alert call.

```java
String[] getSOAPAdditional(
        String[] names)
```

getSOAPAdditionalByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get the additional information to send with the SOAP alert call. This is a location specific function, any action will operate on the specified location.

```java
String[] getSOAPAdditionalByLocation(
        String location,
        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.

```java
Alerting.Action.Login[] getSOAPAuthentication(
        String[] names)
```

getSOAPAuthenticationByLocation( location, 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. This is a location specific function, any action will operate on the specified location.

```java
Alerting.Action.Login[] getSOAPAuthenticationByLocation(
        String location,
        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.

```java
String[] getSOAPProxy(
        String[] names)
```

getSOAPProxyByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get the server the SOAP event call will be made to for each of the specified SOAP events. This is a location specific function, any action will operate on the specified location.

```java
String[] getSOAPProxyByLocation(
        String location,
        String[] names)
```
getScriptArguments( names ) throws InvalidOperation, ObjectDoesNotExist

Gets all arguments for the specified script actions. The actions specified must be of type ‘program’.

```java
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’.

```java
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’.

```java
String[] getSyslogHost(
    String[] names
)
```

getSyslogHostByLocation( location, 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’. This is a location specific function, any action will operate on the specified location.

```java
String[] getSyslogHostByLocation(
    String location
    String[] names
)
```

getSyslogMessageLenLimit( names ) throws InvalidOperation, ObjectDoesNotExist

Get syslog message length limit.

```java
Unsigned Integer[] getSyslogMessageLenLimit(
    String[] names
)
```

getSyslogMessageLenLimitByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get syslog message length limit. This is a location specific function, any action will operate on the specified location.

```java
Unsigned Integer[] getSyslogMessageLenLimitByLocation(
    String location
    String[] names
)
```

getTimeout( names ) throws InvalidOperation, ObjectDoesNotExist

Get how long an action has to run, in seconds (set to 0 disable timing out).
getTimeout( names ) throws InvalidOperation, ObjectDoesNotExist
Get how long an action has to run, in seconds (set to 0 disable timing out). This is a location specific function, any action will operate on the specified location.

getTimeoutByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist
Unsigned Integer[] getTimeoutByLocation(
    String location
    String[] names
)

getTrapCommunity( names ) throws InvalidOperation, ObjectDoesNotExist
Get the SNMP community string for the SNMP trap. The actions specified must be of type 'trap'.

getTrapCommunityByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist
String[] getTrapCommunityByLocation(
    String location
    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'.

getTrapHostByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist
String[] getTrapHostByLocation(
    String location
    String[] names
)

getVerbose( names ) throws InvalidOperation, ObjectDoesNotExist
Get if verbose logging is enabled for this action.

Boolean[] getVerbose(}
getVerboseByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist

Get if verbose logging is enabled for this action. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getVerboseByLocation(
    String location
    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’.

```java
void removeSOAPAuthentication(
    String[] names
)
```

removeSOAPAuthenticationByLocation( location, names ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Disables using HTTP basic authentication with the SOAP Call. The actions specified must be of type ‘soap’. This is a location specific function, any action will operate on the specified location.

```java
void removeSOAPAuthenticationByLocation(
    String location
    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’.

```java
void removeScriptArguments(
    String[] names
)
```

renameAction( names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidOperation

Rename each of the named actions.

```java
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
)

setEmailRecipientsByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the address the alert emails are sent from. This is a location specific function, any action will operate on the specified location.

void setEmailRecipientsByLocation(
    String location
    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
)

setEmailSMTPServerByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SMTP server used to send alert emails for the specified actions. This is a location specific function, any action will operate on the specified location.

void setEmailSMTPServerByLocation(
    String location
    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
)

setEmailSenderByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the specified email addresses to the recipient list for the specified actions. This is a location specific function, any action will operate on the specified location.

void setEmailSenderByLocation(
    String location
    String[] names
    String[] values
)
Alerting.Action Function Reference

**setLogFilePath( names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the file this action logs to.

```java
void setLogFilePath(
    String[] names
    String[] values
)
```

**setLogFilePathByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the file this action logs to. This is a location specific function, any action will operate on the specified location.

```java
void setLogFilePathByLocation(
    String location
    String[] names
    String[] values
)
```

**setSNMPAuthPassword( names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the SNMP password for sending the Notify over SNMPv3. The actions specified must be of type 'trap'.

```java
void setSNMPAuthPassword(
    String[] names
    String[] values
)
```

**setSNMPAuthPasswordByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the SNMP password for sending the Notify over SNMPv3. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

```java
void setSNMPAuthPasswordByLocation(
    String location
    String[] names
    String[] values
)
```

**setSNMPHashAlg( names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the SNMP hash algorithm for sending the Notify over SNMPv3. Valid values are "md5" and "sha1". The actions specified must be of type 'trap'.

```java
void setSNMPHashAlg(
    String[] names
    Alerting.Action.SNMPHashAlg[] values
)
```
setSNMPHashAlgByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SNMP hash algorithm for sending the Notify over SNMPv3. Valid values are "md5" and "sha1". The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

```java
void setSNMPHashAlgByLocation(
    String location
    String[] names
    Alerting.Action.SNMPHashAlg[] values
)
```

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

Set the SNMP encryption key to encrypt SNMPv3 Notify messages. The actions specified must be of type 'trap'.

```java
void setSNMPPrivPassword(
    String[] names
    String[] values
)
```

setSNMPPrivPasswordByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SNMP encryption key to encrypt SNMPv3 Notify messages. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

```java
void setSNMPPrivPasswordByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set the SNMP username for sending the Notify over SNMPv3. The actions specified must be of type 'trap'.

```java
void setSNMPUsername(
    String[] names
    String[] values
)
```

setSNMPUsernameByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set the SNMP username for sending the Notify over SNMPv3. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

```java
void setSNMPUsernameByLocation(
    String location
    String[] names
    String[] values
)
**setSNMPVersion( names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the SNMP version used to send the trap/notify. The actions specified must be of type 'trap'.

```java
void setSNMPVersion{
    String[] names
    Alerting.Action.SNMPVersion[] values
}
```

**setSNMPVersionByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the SNMP version used to send the trap/notify. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

```java
void setSNMPVersionByLocation{
    String location
    String[] names
    Alerting.Action.SNMPVersion[] values
}
```

**setSOAPAdditional( names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the additional information to send with the SOAP alert call.

```java
void setSOAPAdditional{
    String[] names
    String[] values
}
```

**setSOAPAdditionalByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the additional information to send with the SOAP alert call. This is a location specific function, any action will operate on the specified location.

```java
void setSOAPAdditionalByLocation{
    String location
    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'.

```java
void setSOAPAuthentication{
    String[] names
    Alerting.Action.Login[] credentials
}
```
Set the username and password to use to log in with HTTP basic authentication. The actions specified must be of type 'soap'. This is a location specific function, any action will operate on the specified location.

```java
void setSOAPAuthenticationByLocation(
    String location,
    String[] names,
    Alerting.Action.Login[] credentials
)
```

Set the server the SOAP event call will be made to for each of the specified SOAP events.

```java
void setSOAPProxy(
    String[] names,
    String[] values
)
```

Set the server the SOAP event call will be made to for each of the specified SOAP events. This is a location specific function, any action will operate on the specified location.

```java
void setSOAPProxyByLocation(
    String location,
    String[] names,
    String[] values
)
```

Set the program to run including its command line arguments. The actions specified must be of type 'program'.

```java
void setScriptProgram(
    String[] names,
    String[] values
)
```

Set the host to send syslog messages to (if empty, messages will be sent to localhost). The actions specified must be of type 'syslog'.

```java
void setSyslogHost(
    String[] names,
    String[] values
)
```
setSyslogHostByLocation( location, 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'. This is a location specific function, any action will operate on the specified location.

```java
void setSyslogHostByLocation(
    String location
    String[] names
    String[] values
)
```

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

Set syslog message length limit.

```java
void setSyslogMessageLenLimit(
    String[] names
    Unsigned Integer[] values
)
```

setSyslogMessageLenLimitByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set syslog message length limit. This is a location specific function, any action will operate on the specified location.

```java
void setSyslogMessageLenLimitByLocation(
    String location
    String[] names
    Unsigned Integer[] 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).

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

setTimeoutByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set how long an action has to run, in seconds (set to 0 disable timing out). This is a location specific function, any action will operate on the specified location.

```java
void setTimeoutByLocation(
    String location
    String[] names
    Unsigned Integer[] values
)
```
Function Reference

Alerting.Action

**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'.

```java
void setTrapCommunity(
    String[] names
    String[] values
)
```

**setTrapCommunityByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set the SNMP community string for the SNMP trap. The actions specified must be of type 'trap'. This is a location specific function, any action will operate on the specified location.

```java
void setTrapCommunityByLocation(
    String location
    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'.

```java
void setTrapHost(
    String[] names
    String[] values
)
```

**setTrapHostByLocation( location, 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'. This is a location specific function, any action will operate on the specified location.

```java
void setTrapHostByLocation(
    String location
    String[] names
    String[] values
)
```

**setVerbose( names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError**

Set if verbose logging is enabled for this action.

```java
void setVerbose(
    String[] names
    Boolean[] values
)
```
Alerting.Action

setVerboseByLocation( location, names, values ) throws InvalidOperation, ObjectDoesNotExist, InvalidInput, DeploymentError

Set if verbose logging is enabled for this action. This is a location specific function, any action will operate on the specified location.

```java
void setVerboseByLocation(
    String location
    String[] names
    Boolean[] values
)
```

testAction( names ) throws ObjectDoesNotExist

Sends a test event to the named actions to confirm that they are working as expected.

```java
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'.

```java
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.

```java
void uploadActionProgram(
    String name
    Binary Data content
)
```

**Structures**

Alerting.Action.Argument

An argument that is added to the command line when the script is run

```java
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

```c
struct Alerting.Action.Login {
    # The username for basic SOAP authentication
    String username;
    # The username for basic SOAP authentication
    String password;
}
```

Enumerations

Alerting.Action.SNMPHashAlg

```c
class Alerting.Action.SNMPHashAlg {
    # MD5
    md5,
    # SHA-1
    sha1
}
```

Alerting.Action.SNMPVersion

```c
class Alerting.Action.SNMPVersion {
    # SNMPv1
    snmpv1,
    # SNMPv2c
    snmpv2c,
    # SNMPv3
    snmpv3
}
```

Alerting.Action.Type
```c
class 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
```
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.

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.

```java
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
)
```

Structures

AlertCallback.Object

Information on an object that triggered this event.

```java
struct AlertCallback.Object {
    # The type of the object
    AlertCallback.ObjectType type;

    # The name of the object.
    String name;
}
```

Enumerations

AlertCallback.ObjectType

```java
enum AlertCallback.ObjectType {
    # An unexpected type
    Unknown,

    # Actions
    actions,

    # Web Accelerator Profiles
```
aptimizer/profiles,

# Application Scopes
aptimizer/scopes,

# Authenticators
auth,

# Bandwidth Classes
bandwidth,

# Cloud Credentials
cloudcredentials,

# Configuration Files
config,

# DNS Lookup
dns,

# Built-in DNS servers
dnsserver,

# DNS Server Zones
dnsserver/zones,

# 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,

# Locations
locations,

# Log File Export
log_export,

# Monitors
monitors,

# Nodes
nodes,
# Session Persistence Classes
persistence,

# Processes
pids,

# Pools
pools,

# Service Protection Classes
protection,

# Rate Classes
rate,

# RTSP Events
rtsp,

# Rules
rules,

# GLB Services
services,
    servlet,

# Java Servlets
    servlets,

# SIP Events
tsip,

# SLM Classes
    slm,

# SMTP Events
    smtp,
    ssl/cas,
    ssl/client_keys,
    ssl/client_keys/private,
    ssl/client_keys/public,
    ssl/client_keys/request,

# DNSSEC Zone Signing Keys
    ssl/dnssec_keys,
    ssl/server_keys,
    ssl/server_keys/private,
    ssl/server_keys/public,
    ssl/server_keys/request,

# SSL Hardware
    ssl/hw,

# SIP/RTSP
AlertCallback

```
toList,
# 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 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,

    # Cloud Credentials - A cloud API process querying changes to cloud instances
    # is hanging
    cloudcredentials_apistatusprocesshanging,

    # Cloud Credentials - An API call made by the autoscaler process has returned
    # a response that could not be parsed
    cloudcredentials_autoscalerresponseparseerror,

    # Cloud Credentials - An API call made by the autoscaler process has reported
    # an error
    cloudcredentials_autoscalestatusupdateerror,

    # Cloud Credentials - A cloud API process has timed out
    cloudcredentials_autoscalingprocessst timedout,
```
# Cloud Credentials - A Cloud Credentials object has been deleted but it was still in use
cloudcredentials_usedcredsdeleted,

# 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 - Automatic failback after delay
faulttolerance autofailbackafterdelay,

# Fault Tolerance - Auto-failback delay timer cancelled
faulttolerance autofailbacktimercancelled,

# Fault Tolerance - Auto-failback wait period started
faulttolerance autofailbacktimerstarted,

# Fault Tolerance - Auto-failback delay timer stopped due to system failure
faulttolerance autofailbacktimerstopped,

# Fault Tolerance - Some of the BGP neighbors do not have established sessions
faulttolerance bgpneighborsdegraded,

# Fault Tolerance - None of the BGP neighbors have an established session
faulttolerance bgpneighborsfailed,

# Fault Tolerance - There are established sessions with all BGP neighbors
faulttolerance bgpneighborsok,

# Fault Tolerance - There are no valid BGP neighbors defined
faulttolerance bgpnoneighbors,

# Fault Tolerance - The system clock jumped forwards or backwards by more than one second
faulttolerance clockjump,

# Fault Tolerance - The monotonic system clock went backwards
faulttolerance clocknotmonotonic,

# Fault Tolerance - Removing EC2 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 IP Address; local machine is working
faulttolerance_ec2flipperraiselocalworking,

# Fault Tolerance - Moving EC2 IP Address; other machines have failed
faulttolerance_ec2flipperraiseothersdead,

# Fault Tolerance - Problem occurred when managing an EC2 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 - Cannot raise Elastic IP on this machine as no suitable secondary IP is available on the allowed network card(s)
faulttolerance_ec2nosecondaryprivateip,

# 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 - This Traffic Manager has re-raised traffic IP addresses as the remote machine which was hosting them has dropped them
faulttolerance_flipper raiseremotedropped,

# 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_machinererecovered,
AlertCallback Function Reference

# 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 - Some of the monitored OSPF neighbors are not peered
faulttolerance_ospfneighborsdegraded,

# Fault Tolerance - None of the monitored OSPF neighbors are peered
faulttolerance_ospfneighborsfailed,

# Fault Tolerance - All monitored OSPF neighbors are peered
faulttolerance_ospfneighborsok,

# 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 - Failed to send ping packets
faulttolerance_pingsendfail,

# Fault Tolerance - Routing software had a major failure and will be
# restarted
faulttolerance_routingswfailed,

# Fault Tolerance - Routing software has failed and reached its failure limit
faulttolerance_routingswfailurelimitreached,

# Fault Tolerance - Routing software is now operational
faulttolerance_routingswoperational,

# Fault Tolerance - Routing software failed to start
faulttolerance_routingswstartfailed,

# 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_zclustermodderr,

# General - Application firewall control command failed
general_appfirewallcontrolerr,

# General - Application firewall restarted
general_appfirewallcontrolrestarted,

# General - Application firewall started
general_appfirewallcontrolstarted,

# General - Application firewall stopped
general_appfirewallcontrolstopped,

# General - Application firewall control command timed out
general_appfirewallcontroltimeout,

# General - Appliance notification
general_appliance,

# General - An audit log event has occurred
general_audit,

# General - An error occurred during user authentication
general_autherror,

# General - A hostname used for DNS-derived Autoscaling doesn't resolve
general_autoscaleresolvefailure,

# General - Autoscaling not permitted by licence key
general_autoscalinglicenseerror,

# General - There was an error communicating with a child process
general_childcommsfail,

# General - Replication of configuration has failed
general_confrepfailed,

# General - Replication of configuration has timed out
general_confreptimeout,

# General - The built-in DNS server has failed to create a DNS record
general_dnszoneretrecord,

# General - The built-in DNS server has failed to parse a DNS zone file
general_dnszoneres,

# General - The built-in DNS server has failed to validate a DNS zone file
general_dnszonevalidate,

# General - Data Plane Acceleration mode diagnostics
general_dpadiagnostics,

# General - Data Plane Acceleration mode error
general_dpaerror,

# General - Traffic manager failed to get the required data from Amazon
# servers
general_ec2dataretrievalfailed,
# General - Traffic manager has now successfully retrieved the required data from Amazon servers
general_ec2dataretrievalsuccessful,

# General - The EC2 instance is now initialized
general_ec2initialized,

# General - Running out of free file descriptors
general_fewfreefds,

# General - FIPS 140-2 cryptographic module initialization failed
general_fipsfailinit,

# General - FIPS 140-2 cryptographic module operations failed
general_fipsfailops,

# General - Traffic manager failed to get the required data from GCE instance
general_gcedataretrievalfailed,

# General - Traffic manager has now successfully retrieved the required data from GCE instance
general_gcedataretrievalsuccessful,

# General - Failed to load geolocation data
general_geodataloadfail,

# General - A location has been disabled because you have exceeded the licence limit
general_licensetoomanylocations,

# General - Log disk partition full
general_logdiskfull,

# General - Log disk partition usage has exceeded threshold
general_logdiskoverload,

# General - DNS-derived Autoscaling will resume updating, as the DNS server is now responding
general_nameserveravailable,

# General - DNS-derived Autoscaling will not update, as the DNS server is unavailable
general_nameserverunavailable,

# General - Total number of locations exceeded the maximum limit
general_numlocations-exceeded,

# General - Total number of nodes exceeded the maximum number of nodes that can be monitored
general_numnodes-exceeded,

# General - Total number of pools exceeded the maximum limit
general_numpools-exceeded,

# General - Total number of traffic IP group exceeded the maximum limit
general_numtipg-exceeded,

# General - OCSP request (for OCSP stapling) failed
general_ocspstaplingfail,

# General - Insufficient memory for OCSP stapling
general_ocspstaplingnomem,
# General - An OCSP request (for OCSP stapling) reported that a certificate
# was revoked
general_ocspstaplingrevoked,

# General - An OCSP request (for OCSP stapling) reported that a certificate
# was unknown
general_ocspstaplingunknown,

# General - An old but good OCSP response was returned for a revoked
certificate
general_ocspstaplingunrevoked,

# General - Software must be restarted to apply configuration changes
general_restartrequired,

# General - Software is running
general_running,

# General - CRL does not fit in the configured amount of shared memory,
# increase ssl!crl_mem!size and restart software
general_sslcrltoobig,

# General - Virtual Traffic Manager Appliance reboot required
general_mysctlreboot,

# General - Time has been moved back
general_timemovedback,

# General - Virtual Traffic Manager restart/reboot required
general_unspecifriedreboot,

# General - Virtual Traffic Manager Appliance reboot required
general_upgradereboot,

# General - Virtual Traffic Manager software restart required
general_upgraderestart,

# General - The number of simultaneously active connections has reached a
# level that the software cannot process in due time because of CPU
# starvation; there is a high risk of connections timing out
general_zxtmcpustarvation,

# General - The number of simultaneously active connections has reached a
# level that the software cannot process in due time; there is a high risk of
# connections timing out
general_zxtmhightload,

# General - Internal software error
general_zxtmswerror,

# Java - Java runner died
java_javadied,

# Java - Cannot start Java runner, program not found
java_javanotfound,

# Java - Java runner started
java_javastarted,

# Java - Java runner failed to start
java_javastartfail,

# Java - Java support has stopped
java_javastop,
# Java - Java runner failed to terminate
java_javaterminatelfail,

# Java - Servlet encountered an error
java_servleterror,

# License Keys - Analytics Export support has been disabled
licensekeys_analyticsexportlicensedisabled,

# License Keys - Analytics Export support has been enabled
licensekeys_analyticsexportlicenseenabled,

# License Keys - Realtime Analytics support has been disabled
licensekeys_analyticslicensedisabled,

# License Keys - Realtime Analytics support has been enabled
licensekeys_analyticslicenseenabled,

# License Keys - Autoscaling support has been disabled
licensekeys_autoscalingleicensedisabled,

# License Keys - Autoscaling support has been enabled
licensekeys_autoscalingleicenseenabled,

# 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 7 days
licensekeys_expiresoon,

# License Keys - License key expires within 15 days
licensekeys_expiresoon15,

# License Keys - License key expires within 30 days
licensekeys_expiresoon30,

# License Keys - License key expires within 60 days
licensekeys_expiresoon60,

# License Keys - License key expires within 90 days
licensekeys_expiresoon90,

# License Keys - License allows less memory for caching
licensekeys_lessmemallowed,

# License Keys - License key authorized
licensekeys_license-authorized,

# License Keys - License key authorized by authorization code
licensekeys_license-authorized-ts,

# License Keys - License key explicitly disabled from authorization code
licensekeys_license-explicitlydisabled-ts,

# License Keys - Unable to authorize license key
licensekeys_license-graceperiodexpired,
# License Keys - Unable to authorize license key
licensekeys_license-graceperiodexpired-ts,

# License Keys - License server rejected license key; key remains authorized
licensekeys_license-rejected-authorized,

# License Keys - License key rejected from authorization code; key remains authorized
licensekeys_license-rejected-authorized-ts,

# License Keys - License server rejected license key; key is not authorized
licensekeys_license-rejected-unauthorized,

# License Keys - License key rejected from authorization code
licensekeys_license-rejected-unauthorized-ts,

# License Keys - Unable to contact license server; license key remains authorized
licensekeys_license-timedout-authorized,

# License Keys - Unable to run authorization code to completion; key remains valid
licensekeys_license-timedout-authorized-ts,

# License Keys - Unable to contact license server; license key is not authorized
licensekeys_license-timedout-unauthorized,

# License Keys - Unable to run authorization code to completion
licensekeys_license-timedout-unauthorized-ts,

# 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 - Error detected in LicenseStateFile format
licensekeys_licensestate-malformed,

# License Keys - Unable to preserve license state
licensekeys_licensestate-write-failed,

# 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,
# Locations
- Location is now available for GLB Service
  locations_locationavailable,

- Location has been disabled for GLB Service
  locations_locationdisabled,

- Location is being drained for GLB Service
  locations_locationdraining,

- Location has just been enabled for GLB Service
  locations_locationenabled,

- Location has failed for GLB Service
  locations_locationfail,

- A monitor has detected a failure in this location
  locations_locationmonitorfail,

- A monitor has indicated this location is now working
  locations_locationmonitorok,

- Location is not being drained for GLB Service
  locations_locationnotdraining,

- Location is now healthy for GLB Service
  locations_locationok,

- An external SOAP agent has detected a failure in this location
  locations_locationsoapfail,

- An external SOAP agent indicates this location is now working
  locations_locationsoapok,

- Location has become unavailable for GLB Service
  locations_locationunavailable,

- Location no longer contains any machines
  locations_locempty,

- Machine now in location
  locations_locmovemachine,

- Monitor has detected a failure
  monitors_monitorfail,

- Monitor is working
  monitors_monitorok,

- API change process still running after refractory period is over
  pools_apichangeprocesshanging,

- The creation of a new node requested by an autoscaled pool is now complete
  pools_autonodecreationcomplete,

- Creation of new node instigated
  pools_autonodecreationstarted,

- A cloud API call to destroy a node has been started
  pools_autonodedestroyed,

- The destruction of a node in an autoscaled pool is now complete
  pools_autonodedestructioncomplete,
# Pools - A node in an autoscaled pool has disappeared from the cloud
pools_autonodedisappeared,

# Pools - IP address of newly created instance already existed in pool's node
# list
pools_autonodeexisted,

# Pools - Node has no public IP address
pools_autonodenopublicip,

# Pools - A node in a DNS-derived autoscaled pool has been removed
pools_autonoderemoved,

# Pools - The status of a node in an autoscaled pool has changed
pools_autonodestatuschange,

# Pools - Two pools are trying to use the same instance
pools_autoscalednodecontested,

# Pools - An autoscaled pool is now refractory
pools_autoscaledpoolrefractory,

# Pools - Over maximum size - shrinking
pools_autoscaledpooltoobig,

# Pools - Minimum size undercut - growing
pools_autoscaledpooltoosmall,

# Pools - The ‘imageid’ was empty when attempting to create a node in an
# autoscaled pool
pools_autoscaleinvalidargforcreatenode,

# Pools - ‘unique id’ was empty when attempting to destroy a node in an
# autoscaled pool
pools_autoscaleinvalidargfordeletenode,

# Pools - A pool config file has been updated by the autoscaler process
pools_autoscalepoolconfupdate,

# Pools - A node created by the autoscaler has the wrong imageid
pools_autoscalewrongimageid,

# Pools - A node created by the autoscaler has a non-matching name
pools_autoscalewrongname,

# Pools - A node created by the autoscaler has the wrong sizeid
pools_autoscalewrongsized,

# Pools - An API process that should have created or destroyed a node has
# failed to produce the expected result
pools_autoscalingchangeprocessfailure,

# Pools - Autoscaling for a pool has been disabled due to errors
# communicating with the cloud API
pools_autoscalingdisabled,

# Pools - Minimum size reached, cannot shrink further
pools_autoscalinghitfloor,

# Pools - Maximum size reached by autoscaled pool, cannot grow further
pools_autoscalinghitroof,

# Pools - An autoscaled pool is waiting to grow
pools_autoscalinghysteresiscantgrow,
# Pools - An autoscaled pool is waiting to shrink
pools_autoscalinghysteresiscantshrink,

# Pools - An autoscaled pool's state has changed
pools_autoscalingpoolstatechange,

# Pools - An autoscaled pool has failed completely
pools_autoscalingresuscitaeppool,

# Pools - HTTP response contained an invalid Content-Length header
pools_badcontentlen,

# Pools - Attempt to scale down a pool that only had pending nodes or none at all
pools_cannotshrinkemptypool,

# Pools - Node returned invalid EHLO response
pools_ehloinvalid,

# Pools - Removed node is in use and will be drained
pools_nodedrainingtodelete,

# Pools - Draining to delete period for node has expired
pools_nodedrainingtodeletetimeout,

# 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
pools_protection_summary,

# Rules - Rule attempted to use Web Accelerator but it is not enabled
rules_aptimizedisabled,

# Rules - Rule selected an unknown Web Accelerator profile
rules_optimizeunknownprofile,

# Rules - Rule selected an unknown Web Accelerator scope
rules_optimizeuseunknownscope,

# Rules - data.local.set() has run out of space
rules_datalocalstorefull,

# 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_rulelogmsgserror,

# Rules - Rule logged a warning message using log.warn
rules_rulelogmsgwarn,

# Rules - Rule selected an unknown session persistence class
rules_rulenopersistence,

# Rules - Rule exceeded execution time warning threshold
rules_ruleoverrun,

# 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,

# Rules - GLB service rule aborted during execution
rules_serviceruleabort,

# Rules - GLB service rule specified a location that has either failed or been marked as draining in the service configuration
rules_servicerulelocdead,

# Rules - GLB service rule specified a location that is not configured for the service
rules_servicerulelocnotconfigured,

# Rules - GLB service rule specified an unknown location
rules_servicerulelocunknown,

# GLB Services - Active datacentre mismatches among cluster members
services_glbactivedcmismatch,

# GLB Services - A DNS Query returned IP addresses that are not configured for any location that is currently alive
services_glbdeadlocmissingips,

# GLB Services - Failed to alter DNS packet for global load balancing
services_glbfailalter,

# GLB Services - Failed to write log file for GLB service
services_glblogwritefail,

# GLB Services - Manual failback triggered
services_glbmanualfailback,

# GLB Services - A DNS Query returned IP addresses that are not configured in any location
services_glbmissingips,

# GLB Services - A location has been set as active for a GLB service
services_glbnewmaster,

# GLB Services - No valid location could be chosen for Global Load Balancing
services_glbnolocations,

# GLB Services - GLB Service has failed
services_glbservicedied,

# GLB Services - GLB Service has recovered
services_glbserviceok,

# GLB Services - There are too many Data Centers configured and the Global Load Balancing feature is not guaranteed to work reliably with more than 255 Data Centres
services_glbtoomanylocations,
# SLM Classes - SLM shared memory 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 Brocade vTM Administration Server.
testaction,

# Virtual Servers - A protocol error has occurred
vserver_connerror,

# Virtual Servers - A socket connection failure has occurred
vserver_connfail,

# Virtual Servers - The built-in DNS server has successfully added a DNS zone
vserver_dnsaddzone,

# Virtual Servers - The built-in DNS server has failed to add a DNS zone
vserver_dnserroraddzone,

# Virtual Servers - The built-in DNS server has failed to delete a DNS zone
vserver_dnserrordeletezone,

# Virtual Servers - DNSSEC zone contains expired signatures
vserver_dnssecexpired,

# Virtual Servers - DNSSEC zone contains signatures that are about to expire
vserver_dnssecexpires,

# Virtual Servers - DNS zone has been deleted
vserver_dnszonedelete,

# Virtual Servers - A virtual server request log file was deleted (appliances only)
vserver_logfiledeleted,

# Virtual Servers - Dropped connection, request exceeded max_client_buffer limit
vserver_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 (hardware available)
vservers_privkeyok,

# Virtual Servers - Error compressing HTTP response
vservers_respcmpfail,

# Virtual Servers - Response headers from webserver too large
vservers_responsetoolarge,

# Virtual Servers - No suitable ports available for streaming data connection
vservers_rtpstreamnoports,

# 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 - SSL handshake messages have exceeded the size permitted
# by configuration
vservers_sslhandshakemsgsizelimit,

# Virtual Servers - SSL re-handshake requests have exceeded the frequency
# permitted by configuration
vservers_sslrehandshakemininterval,

# Virtual Servers - Certificate Authority certificate expired
vservers_vscacertexpired,

# Virtual Servers - Certificate Authority certificate will expire within
# seven days
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_vsslcertexpired,

# Virtual Servers - Public SSL certificate will expire within seven days
vservers_vsslcerttoexpire,

# Virtual Servers - Virtual server started
vservers_vstart,

# Virtual Servers - Virtual server stopped
vservers_vstop,

# Traffic Managers - Configuration update refused: traffic manager version
# mismatch
zxtms_versionmismatch
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 Brocade Virtual Traffic Manager Appliance. This interface is only available on an appliance and is deprecated; use the System.RequestLogs interface instead.

Methods

deleAllVSAccessLogs() 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 
 )

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;

## System.Cache

URI: http://soap.zeus.com/zxtm/1.3/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.

### 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.

```java
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.

```java
void clearMatchingCacheContent(
    System.Cache.Protocol protocol
    String host_wildcard
    String path_wildcard
)
```

#### clearWebCache()

Clear all entries from the Web Cache for this machine.

```java
void clearWebCache()
```

#### getCacheContent( protocol, host_wildcard, path_wildcard, max_entries )

Get information about the Web Cache entries matching the input arguments.

```java
System.Cache.CacheContentInfo getCacheContent(
    System.Cache.Protocol protocol
    String host_wildcard
)
```
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.


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.

    # 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;

    # Whether or not Web Accelerator has optimized the content of this cache entry.
    Boolean optimized;

    # 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[] varies;

    # The specific web browsers for which this entry is cached.
    String[] browsers;
}
System.Cache.CacheContentInfo

This structure contains the information about the cache content.

```c
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.

```c
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;
}
```

Enumerations

System.Cache.Protocol

This enumeration defines the possible protocols for cache entries.

```c
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
}
```

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.

**Methods**

**getAllConnections()**

Get a list of all connections, current and recent, for this machine.

```java
System.Connections.Connection[] getAllConnections()
```

**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.

```java
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.
}```
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.

Enumerations

System.Connections.ConnectionState

This enumeration defines the possible states for a particular connection.

class 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
}
Methods

**addLicenseKeys(license_texts) throws ObjectAlreadyExists, InvalidInput**

Create and add each of the named license keys.

```java
Integer[] addLicenseKeys(
    String[] license_texts
)
```

**deleteLicenseKeys(serials) throws ObjectDoesNotExist**

Delete each of the named license keys.

```java
void deleteLicenseKeys(
    Integer[] serials
)
```

**getAllLicenseKeys()**

Get a list of all the serial numbers of the existing license keys.

```java
Integer[] getAllLicenseKeys()
```

**getCurrentLicenseKey()**

Get the serial number of the license key currently being used by this machine.

```java
Integer getCurrentLicenseKey()
```

**getLicenseKeys(serials) throws ObjectDoesNotExist**

For each of the named license keys, get the license key structure.

```java
System.LicenseKeys.LicenseKey[] getLicenseKeys(
    Integer[] serials
)
```

**getRawLicenseKeys(serials) throws ObjectDoesNotExist**

For each of the named license keys, get the raw license key text.

```java
String[] getRawLicenseKeys(
    Integer[] serials
)
```

Structures

**System.LicenseKeys.LicenseKey**

This structure contains the basic information for a license key. It is used when adding, deleting or retrieving license keys.

```java
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 Brocade vTM 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;
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.

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
)

ergErrorLogLines( 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
)

ergErrorLogString()
Get the error log for this machine as a string.

String getErrorLogString()

Structures

System.Log.AccessDenied

This is the operation parameters structure for ‘accessdenied’ operations (host denied by access restrictions). It is used when getting Audit Log data.

    # A host value.
    String host;
}
System.Log.AddAuthenticator

This is the operation parameters structure for 'addauth' operations (authenticator added). It is used when getting Audit Log data.

```java
    # An authenticator being modified.
    String modauth;
    # Type of an authenticator being modified.
    String authtype;
}
```

System.Log.AddFile

This is the operation parameters structure for 'addfile' operations (file added). It is used when getting Audit Log data.

```java
    # A file on the filesystem being modified.
    String file;
}
```

System.Log.AddGroup

This is the operation parameters structure for 'addgroup' operations (group added). It is used when getting Audit Log data.

```java
    # A group being modified.
    String modgroup;
}
```

System.Log.AddUser

This is the operation parameters structure for 'adduser' operations (user added). It is used when getting Audit Log data.

```java
    # A user being modified.
    String moduser;
    # A group being modified.
    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.

```java
    # Arbitrary text.
    String text;
    # An arbitrary object.
    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.

```csharp
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 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.

```csharp
    # An authenticator being modified.
    String modauth;
    # An authenticator that was copied.
    String oldauth;
    # Type of an authenticator being modified.
    String authtype;
}
```

System.Log.CopyFile

This is the operation parameters structure for 'copyfile' operations (file copied). It is used when getting Audit Log data.

```csharp
    # A file that was copied or renamed.
    String oldfile;
    # A file on the filesystem being modified.
    String file;
}
```
System.Log.CopyGroup

This is the operation parameters structure for 'copygroup' operations (group copied). It is used when getting Audit Log data.

```java
    # A group being modified.
    String modgroup;
    # A group that was copied.
    String oldgroup;
}
```

System.Log.DeleteAuthenticator

This is the operation parameters structure for 'delauth' operations (authenticator deleted). It is used when getting Audit Log data.

```java
    # An authenticator being modified.
    String modauth;
    # Type of an authenticator being modified.
    String authtype;
}
```

System.Log.DeleteFile

This is the operation parameters structure for 'delfile' operations (file deleted). It is used when getting Audit Log data.

```java
    # A file on the filesystem being modified.
    String file;
}
```

System.Log.DeleteGroup

This is the operation parameters structure for 'delgroup' operations (group deleted). It is used when getting Audit Log data.

```java
    # A group being modified.
    String modgroup;
}
```

System.Log.DeleteUser

This is the operation parameters structure for 'deluser' operations (user deleted). It is used when getting Audit Log data.

```java
    # A user being modified.
    String moduser;
    # A file on the filesystem being modified.
    String file;
}
```
System.Log/Login
This is the operation parameters structure for 'login' operations (logged in). It is used when getting Audit Log data.

```java
struct System.Log>Login implements System.Log.OpParam {
    // A login type, i.e. UI, basicauth, or SSH.
    String logintype;

    // A login timeout value.
    String timeout;
}
```

System.Log/LoginFail
This is the operation parameters structure for 'loginfail' operations (failed login attempt). It is used when getting Audit Log data.

```java
struct System.Log/LoginFail implements System.Log.OpParam {
    // A login type, i.e. UI, basicauth, or SSH.
    String logintype;

    // Resource being accessed.
    String resource;
}
```

System.Log/LoginLimitHit
This is the operation parameters structure for 'loginlockout' operations (user account disabled). It is used when getting Audit Log data.

```java
struct System.Log/LoginLimitHit implements System.Log.OpParam {
    // Arbitrary text.
    String text;
}
```

System.Log/LoginSuspended
This is the operation parameters structure for 'loginsusp' operations (suspended user login attempt). It is used when getting Audit Log data.

```java
struct System.Log/LoginSuspended implements System.Log.OpParam {
    // A login type, i.e. UI, basicauth, or SSH.
    String logintype;
}
```

System.Log/MaintenanceCLICmd
This is the operation parameters structure for 'maintclicmd' operations (maintenance CLI command). It is used when getting Audit Log data.

```java
    // The command being run.
    String cmd;

    // The arguments for the command being run.
    String args;
}
```
System.Log.ModifyFile
This is the operation parameters structure for 'filemod' operations (file modified). It is used when getting Audit Log data.

```plaintext
    # A file on the filesystem being modified.
    String file;
}
```

System.Log.ModifyKey
This is the operation parameters structure for 'keymod' operations (config modified). It is used when getting Audit Log data.

```plaintext
    # A configuration key.
    String key;

    # A configuration value.
    String value;

    # A value that was changed.
    String oldval;

    # A file on the filesystem being modified.
    String file;
}
```

System.Log.ModifyRule
This is the operation parameters structure for 'rulemod' operations (modified rule). It is used when getting Audit Log data.

```plaintext
    # A file on the filesystem being 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.

```plaintext
    # A user being modified.
    String moduser;
}
```

System.Log.NoAccessPermission
This is the operation parameters structure for 'noperm' operations (user was refused permission whilst accessing section/item). It is used when getting Audit Log data.

```plaintext
    # A section.
    String sec;
}
```
System.Log.NoChangePermission
This is the operation parameters structure for 'nopostperm' operations (user was refused permission to update data in section). It is used when getting Audit Log data.

    # A section.
    String sec;
}

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.PasswordExpired
This is the operation parameters structure for 'passwordexpired' operations (user's password has expired.). It is used when getting Audit Log data.

    # A login type, i.e. UI, basicauth, or SSH.
    String logintype;
    # Resource being accessed.
    String resource;
}

System.Log.RegenerateUUID
This is the operation parameters structure for 'uuidregen' operations (user regenerated UUID.). It is used when getting Audit Log data.

    # A configuration value.
    String value;
    # A value that was changed.
    String oldval;
    # Arbitrary text.
    String text;
}

System.Log.RemoveKey
This is the operation parameters structure for 'removekey' operations (removed config key). It is used when getting Audit Log data.

    # A configuration key.
    String key;
    # A value that was changed.
    String oldval;
    # A file on the filesystem being modified.
    String file;
}
**System.Log.RenameFile**

This is the operation parameters structure for 'renfile' operations (file renamed). It is used when getting Audit Log data.

```java
    # A file that was copied or renamed.
    String oldfile;

    # A file on the filesystem being modified.
    String file;
}
```

**System.Log.SessionTerminated**

This is the operation parameters structure for 'terminated' operations (user session terminated). It is used when getting Audit Log data.

```java
    # Arbitrary text.
    String text;
}
```

**System.Log.StartVS**

This is the operation parameters structure for 'startvs' operations (virtual server started). It is used when getting Audit Log data.

```java
    # A virtual server.
    String vs;
}
```

**System.Log.StopVS**

This is the operation parameters structure for 'stopvs' operations (virtual server stopped). It is used when getting Audit Log data.

```java
    # A virtual server.
    String vs;
}
```

**System.Log.TrafficManagerActivated**

This is the operation parameters structure for 'activated' operations (traffic manager activated). It is used when getting Audit Log data.

```java
    # A host value.
    String host;
}
```

**Enumerations**

**System.Log.OperationType**

This enumeration defines the possible types of operations that may exist in the audit log.

```java
enum System.Log.OperationType {
```
Function Reference

System Log

# 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 TrafficManagerActivated operation occurs when a traffic manager is restored from a pending state after a failure has occurred. This results in the traffic manager's Traffic IPs being restored to it. It appears as an 'activated' operation in the Audit Log.
TrafficManagerActivated,

# An AddAuthenticator operation type occurs when a new authenticator is created. It appears as an 'addauth' operation in the Audit Log.
AddAuthenticator,

# 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,

# An AddGroup operation occurs when a new group of users is created. It appears as an 'addgroup' operation in the Audit Log.
AddGroup,

# An AddUser operation occurs when a new user is added. It appears as an 'adduser' operation in the Audit Log.
AddUser,

# 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,

# The admin user's password has been reset from the system console.
AdminPasswordReset,

# 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,

# 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,

# 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,

# A DeleteAuthenticator operation type occurs when an existing authenticator is deleted. It appears as a 'delauth' operation in the Audit Log.
DeleteAuthenticator,

# 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 DeleteGroup operation occurs when a group of users is deleted. It appears as an 'delgroup' operation in the Audit Log.
DeleteGroup,
System Log Function Reference

# A DeleteUser operation occurs when an existing user is deleted. It appears as an 'deluser' operation in the Audit Log.
DeleteUser,

# 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 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 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 LoginLimitHit operation occurs when the limit on login attempts is hit for a particular user. It appears as an 'loginlockout' operation in the Audit Log.
LoginLimitHit,

# A LoginSuspended operation occurs when suspended user attempts to login. It appears as an 'loginsusp' operation in the Audit Log.
LoginSuspended,

# 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 MaintenanceCLICmd operation occurs when a command is run in the appliance maintenance CLI. It appears as an 'maintclicmd' operation in the Audit Log.
MaintenanceCLICmd,

# A NoAccessPermission operation occurs when a user is refused permission whilst 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 PasswordExpired operation occurs when a user's password is too old and expires. It appears as an 'passwordexpired' operation in the Audit Log.
PasswordExpired,

# A SystemSettingsReapplied operation occurs when operating system configuration is reapplied on an appliance. It appears as an 'reapplynetwork' operation in the Audit Log.
SystemSettingsReapplied,
# A RemoveKey operation type occurs when a key is removed from a config file, usually because a key is being made location specific. It appears as a 'removekey' operation in the Audit Log.
RemoveKey,

# 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,

# No recent activity has been seen for this user on the REST API.
RestSessionEnd,

# An authenticated user has accessed the REST API
RestSessionStart,

# A ConfigRefreshed operation occurs when the configuration is forcibly reloaded. It appears as an 'revalidate' operation in the Audit Log.
ConfigRefreshed,

# 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 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,

# A SuspensionExpired operation occurs when a users suspension expires, restoring them to active status. This usually occurs when after too many attempts have been made to login to an account. It appears as an 'suspensionexpired' operation in the Audit Log.
SuspensionExpired,

# A Synchronise operation type occurs when configuration is replicated from one machine across the cluster, for example in order to resolve a conflict arising from one machine being unavailable at the time when a configuration change was made. It appears as a 'synchronise' operation in the Audit Log.
Synchronise,

# A SessionTerminated operation occurs when a users session is terminated externally. It appears as an 'terminated' operation in the Audit Log.
SessionTerminated,

# 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 ModifyUser operation occurs when an existing user is modified. It appears as an 'usermod' operation in the Audit Log.
ModifyUser,

# UUID has been regenerated by the user.
RegenerateUUID
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.

Methods

getAllClusterMachines()

Gets all of the machines in this traffic manager’s cluster.

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()

getStingrayOSVersion()

Get the Operating System version for this appliance.
String getStingrayOSVersion()

getTrafficManagerUptime()

Get the time (in seconds) that the traffic manager has been running for.
Unsigned Integer getTrafficManagerUptime()

getUUID()

Get the Universally Unique Identifier (UUID) for this traffic manager.
String getUUID()

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()

Structures

This structure contains information about a traffic manager in the cluster.

    # 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;
}

System.NAT
URI: http://soap.zeus.com/zxtm/1.0/System/NAT/
The NAT interface allows management of custom NAT rules. Using this interface, you can create, delete and view custom NAT rules.

Methods

addManyToOneAllPorts( all_ports_data ) throws InvalidInput, ObjectDoesNotExist
Add a many-to-one all-ports NAT rule

void addManyToOneAllPorts(
    System.NAT.ManyToOneAllPortsRule[] all_ports_data
)

addManyToOnePortLocked( port_locked_data ) throws InvalidInput, ObjectDoesNotExist
Add a many-to-one port locked NAT rule

void addManyToOnePortLocked(
    System.NAT.ManyToOnePortLockedRule[] port_locked_data
)

addOneToOne( one_to_one_data ) throws InvalidInput, ObjectDoesNotExist
Add a one-to-one NAT rule
void addOneToOne(
    System.NAT.OneToOneRule[] one_to_one_data
)

addPortMapping( port_mapping_data ) throws InvalidInput, ObjectDoesNotExist
Add a port mapping rule for a virtual server
void addPortMapping(
    System.NAT.PortMappingRule[] port_mapping_data
)

getManyToOneAllPortsList()
Get a list of the many-to-one all-ports NAT rules
System.NAT.ManyToOneAllPortsRule[] getManyToOneAllPortsList()

getManyToOnePortLockedList()
Get a list of the many-to-one port-locked NAT rules
System.NAT.ManyToOnePortLockedRule[] getManyToOnePortLockedList()

getOneToOneList()
Get a list of the one-to-one NAT rules
System.NAT.OneToOneRule[] getOneToOneList()

getPortMappingList()
Get a list of the port mapping NAT rules
System.NAT.PortMappingRule[] getPortMappingList()

removeManyToOneAllPorts( all_ports_data ) throws InvalidInput, ObjectDoesNotExist
Remove a many-to-one all-ports rule matching the rule provided
void removeManyToOneAllPorts(
    System.NAT.ManyToOneAllPortsRule[] all_ports_data
)

removeManyToOnePortLocked( port_locked_data ) throws InvalidInput, ObjectDoesNotExist
Remove a many-to-one port-locked rule matching the rule provided
void removeManyToOnePortLocked(
    System.NAT.ManyToOnePortLockedRule[] port_locked_data
)

removeOneToOne( one_to_one ) throws InvalidInput, ObjectDoesNotExist
Remove a one-to-one rule matching the rule provided
void removeOneToOne(
    System.NAT.OneToOneRule[] one_to_one
)
Function Reference

System.NAT

removePortMapping( port_mapping_data ) throws InvalidInput, ObjectDoesNotExist

Remove a port mapping rule matching the rule provided

```java
void removePortMapping(
    System.NAT.PortMappingRule[] port_mapping_data
)
```

Structures

System.NAT.ManyToOneAllPortsRule

A list of config key value pairs for a many-to-one all-ports NAT rule

```java
struct System.NAT.ManyToOneAllPortsRule {
    # The name of the TIP group this rule applies to
    String tipgroup;

    # The name of the pool this rule filters on
    String pool;
}
```

System.NAT.ManyToOnePortLockedRule

A list of config key value pairs for a many-to-one port-locked NAT rule

```java
struct System.NAT.ManyToOnePortLockedRule {
    # The name of the TIP group this rule applies to
    String tipgroup;

    # The name of the pool this rule filters on
    String pool;

    # The protocol this rule applies to (TCP, UDP, etc.)
    String protocol;

    # The port number this rule applies to
    Integer port;
}
```

System.NAT.OneToOneRule

A list of config key value pairs for a One To One NAT rule

```java
struct System.NAT.OneToOneRule {
    # The name of the TIP group this rule applies to
    String tipgroup;

    # The name of the IP address this rule filters on
    String ip;

    # Whether or not this rule has an associated inbound rule
    String inbound;
}
```
System.NAT.PortMappingRule

A list of config key value pairs for a Port Mapping NAT rule

```c
struct System.NAT.PortMappingRule {
    # The name of the virtualserver this rule applies to
    String virtualserver;

    # The first port in the port range the virtual server will now listen on
    Integer first;

    # The last port in the port range the virtual server will now listen on
    Integer last;
}
```

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 Brocade vTM appliance. This interface is only available on an appliance.

Methods

deleteAllVSRequestLogs() throws InvalidOperation

Delete all the request logs for all virtual servers.

```java
void deleteAllVSRequestLogs()
```

deleteVSRequestLog( logfiles ) throws InvalidInput, InvalidOperation

Delete the specified request logs.

```java
void deleteVSRequestLog(
    String[] logfiles
)
```

deleteVSRequestLogs( vservers ) throws InvalidOperation

Delete the request logs for specific virtual servers.

```java
void deleteVSRequestLogs(
    String[] vservers
)
```

getAllVSRequestLogs() throws InvalidOperation

Get the request logs for all virtual servers.

```java
System.RequestLogs.VSRequestLog[] getAllVSRequestLogs()
```

getVSRequestLogs( vservers ) throws InvalidOperation

Get the request logs for specific virtual servers.

```java
System.RequestLogs.VSRequestLog[][] getVSRequestLogs(
    String[] vservers
)
```
Strings[] vservers

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;
}

System.Stats

URI: http://soap.zeus.com/zxtm/1.0/System/Stats/

The System.Stats interface retrieves statistical information about the system. Note: This interface is essentially a SOAP implementation of part of the SNMP interface. If you experience any performance issues using this interface, it is recommended trying SNMP directly.

Methods

getActionNumber()

The number of actions configured in the traffic manager.

Integer getActionNumber()

getActions()

Gets the list of Alerting Actions configured.

Strings[] getActions()

getActionsProcessed( names ) throws InvalidInput, InvalidObjectName

Number of times this action has been processed, for each of the named Actions.

Integer[] getActionsProcessed(
    Strings[] names
)
getAnalyticsTransactionsDropped()
Count of transaction metadata records that have been dropped
Integer getAnalyticsTransactionsDropped()

getAnalyticsTransactionsExported()
Count of transaction metadata records that have been exported
Integer getAnalyticsTransactionsExported()

getAnalyticsTransactionsMemoryUsage()
Number of bytes queued in the transaction export transmit buffers.
Integer getAnalyticsTransactionsMemoryUsage()

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()
getAuthenticatorErrors( names ) throws InvalidInput, InvalidObjectName
Number of connection errors that have occurred when trying to connect to an authentication server, for each of the named Authenticators.

```java
Integer[] getAuthenticatorErrors(
    String[] names
)
```

getAuthenticatorFails( names ) throws InvalidInput, InvalidObjectName
Number of times this Authenticator has failed to authenticate, for each of the named Authenticators.

```java
Integer[] getAuthenticatorFails(
    String[] names
)
```

getAuthenticatorNumber()
The number of Authenticators.

```java
Integer getAuthenticatorNumber()
```

getAuthenticatorPasses( names ) throws InvalidInput, InvalidObjectName
Number of times this Authenticator has successfully authenticated, for each of the named Authenticators.

```java
Integer[] getAuthenticatorPasses(
    String[] names
)
```

getAuthenticatorRequests( names ) throws InvalidInput, InvalidObjectName
Number of times this Authenticator has been asked to authenticate, for each of the named Authenticators.

```java
Integer[] getAuthenticatorRequests(
    String[] names
)
```

getAuthenticators()
Gets the list of Authenticators configured.

```java
String[] getAuthenticators()
```

getBandwidthClassBytesDrop( names ) throws InvalidInput, InvalidObjectName
Bytes dropped by this bandwidth class, for each of the named BandwidthClasses.

```java
Long[] getBandwidthClassBytesDrop(
    String[] names
)
```

getBandwidthClassBytesOut( names ) throws InvalidInput, InvalidObjectName
Bytes output by connections assigned to this bandwidth class, for each of the named BandwidthClasses.

```java
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()

getBandwidthClassPktsDrop( names ) throws InvalidInput, InvalidObjectName

Number of packets dropped by this bandwidth class, for each of the named BandwidthClasses.

Long[] getBandwidthClassPktsDrop(
    String[] names
)

getBandwidthClasses()

Gets the list of Bandwidth Classes configured.

String[] getBandwidthClasses()

getCloudcredentialsClassNumber()

The number of cloud credentials sets defined.

Integer getCloudcredentialsClassNumber()

getCloudcredentialsNodeCreations( names ) throws InvalidInput, InvalidObjectName

The number of instance creation API requests made with this set of cloud credentials, for each of the named Cloudcredentialses.

Integer[] getCloudcredentialsNodeCreations(
    String[] names
)

getCloudcredentialsNodeDeletions( names ) throws InvalidInput, InvalidObjectName

The number of instance destruction API requests made with this set of cloud credentials, for each of the named Cloudcredentialses.

Integer[] getCloudcredentialsNodeDeletions(
    String[] names
)
getCloudcredentialsStatusRequests( names ) throws InvalidInput, InvalidObjectName

The number of status API requests made with this set of cloud credentials, for each of the named Cloudcredentialses.

Integer[] getCloudcredentialsStatusRequests(
    String[] names
)

getCloudcredentialses()

Gets the list of Cloud Credentials configured.

String[] getCloudcredentialses()

gsCoreUtilizationPercent( core_ids ) throws InvalidInput, InvalidObjectName

The cpu utilization of the data plane acceleration core, for each of the specified DpaCoreUtilizations.

Integer[] getCoreUtilizationPercent(
    String[] core_ids
)

dataEntries()

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()

dataPlaneAccelCoreNumber()

The number of data plane acceleration cores.

Integer getDataPlaneAccelCoreNumber()

DpaCoreUtilizations()

Gets the list of Data plane acceleration cores configured.

String[] getDpaCoreUtilizations()

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()
```

**getGlbServiceDiscarded(names)** throws **InvalidInput, InvalidObjectName**

Number of A records this GLB Service has discarded, for each of the named GlbServices.

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

**getGlbServiceNumber()**

The number of GLB Services on this system.

```
Integer getGlbServiceNumber()
```

**getGlbServiceResponses(names)** throws **InvalidInput, InvalidObjectName**

Number of A records this GLB Service has altered, for each of the named GlbServices.

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

**getGlbServiceUnmodified(names)** throws **InvalidInput, InvalidObjectName**

Number of A records this GLB Service has passed through unmodified, for each of the named GlbServices.

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

**getGlbServices()**

Gets the list of GLB services configured.

```
String[] getGlbServices()
```

**getHourlyPeakBytesInPerSecond()**

The peak bytes received from clients per second in the last hour.

```
Integer getHourlyPeakBytesInPerSecond()
```

**getHourlyPeakBytesOutPerSecond()**

The peak bytes sent to clients per second in the last hour.

```
Integer getHourlyPeakBytesOutPerSecond()
```
getHourlyPeakRequestsPerSecond()
The peak requests per second in the last hour.
Integer getHourlyPeakRequestsPerSecond()

getHourlyPeakSSLConnectionsPerSecond()
The peak ssl connections per second in the last hour.
Integer getHourlyPeakSSLConnectionsPerSecond()

getHttp2ConnectionsOpen()
The number of HTTP/2 connections currently open.
Integer getHttp2ConnectionsOpen()

getHttp2HeadersBytesReadCompressed()
Total size of compressed HTTP/2 headers read.
Long getHttp2HeadersBytesReadCompressed()

getHttp2HeadersBytesReadUncompressed()
Total size of uncompressed HTTP/2 headers read.
Long getHttp2HeadersBytesReadUncompressed()

getHttp2StreamsClosedByPeer()
Total number of HTTP/2 streams closed by peer.
Long getHttp2StreamsClosedByPeer()

getHttp2StreamsClosedByUs()
Total number of HTTP/2 streams closed by us.
Long getHttp2StreamsClosedByUs()

getHttp2StreamsOpen()
The number of HTTP/2 streams currently open.
Integer getHttp2StreamsOpen()

getHttp2StreamsPushPromiseSentByPeer()
Total number of HTTP/2 push-promise streams sent by peer.
Long getHttp2StreamsPushPromiseSentByPeer()

getHttp2StreamsPushPromiseSentByUs()
Total number of HTTP/2 push-promise streams sent by us.
Long getHttp2StreamsPushPromiseSentByUs()
**getHttp2StreamsResetByPeer()**

Total number of HTTP/2 streams reset by peer.

```java
Long getHttp2StreamsResetByPeer()
```

**getHttp2StreamsResetByUs()**

Total number of HTTP/2 streams reset by us.

```java
Long getHttp2StreamsResetByUs()
```

**getHttp2StreamsTotalControlBytesRead()**

Total number of HTTP/2 control frame bytes read.

```java
Long getHttp2StreamsTotalControlBytesRead()
```

**getHttp2StreamsTotalControlBytesWritten()**

Total number of HTTP/2 control frame bytes written.

```java
Long getHttp2StreamsTotalControlBytesWritten()
```

**getHttp2StreamsTotalCreated()**

Total number of HTTP/2 streams created.

```java
Long getHttp2StreamsTotalCreated()
```

**getHttp2StreamsTotalDataBytesRead()**

Total number of HTTP/2 data frame bytes read.

```java
Long getHttp2StreamsTotalDataBytesRead()
```

**getHttp2StreamsTotalDataBytesWritten()**

Total number of HTTP/2 data frame bytes written.

```java
Long getHttp2StreamsTotalDataBytesWritten()
```

**getInterfaceCollisions( names ) throws InvalidInput, InvalidObjectName**

The number of collisions reported by this interface, for each of the named Interfaces.

```java
Integer[] getInterfaceCollisions(
    String[] names
)
```

**getInterfaceNumber()**

The number of network interfaces.

```java
Integer getInterfaceNumber()
```

**getInterfaceRxBytes( names ) throws InvalidInput, InvalidObjectName**

Bytes received by this interface, for each of the named Interfaces.

```java
Long[] getInterfaceRxBytes(
    String[] names
)
```
Function Reference

System.Stats

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()  

getListenIPBytesIn( listen_ip_addresses ) throws InvalidInput, InvalidObjectName

Bytes sent to this listening IP, for each of the specified ListenIPs.

Long[] getListenIPBytesIn(
   String[] listen_ip_addresses
)

getListenIPBytesOut( listen_ip_addresses ) throws InvalidInput, InvalidObjectName

Bytes sent from this listening IP, for each of the specified ListenIPs.

Long[] getListenIPBytesOut(
   String[] listen_ip_addresses
)

getListenIPCurrentConn( listen_ip_addresses ) throws InvalidInput, InvalidObjectName

TCP connections currently established to this listening IP, for each of the specified ListenIPs.

Integer[] getListenIPCurrentConn(
   String[] listen_ip_addresses
)

getListenIPMaxConn( listen_ip_addresses ) throws InvalidInput, InvalidObjectName

Maximum number of simultaneous TCP connections this listening IP has processed at any one time, for each of the specified ListenIPs.

Integer[] getListenIPMaxConn(
   String[] listen_ip_addresses
)

getListenIPTotalConn( listen_ip_addresses ) throws InvalidInput, InvalidObjectName

Formerly provided the number of requests sent to this listening IP, now deprecated, for each of the specified ListenIPs.
System.Stats Function Reference

Integer[] getListenIPTotalConn(String[] listen_ip_addresses)

getListenIPTotalRequests(listen_ip_addresses) throws InvalidInput, InvalidObjectName

Requests sent to this listening IP, for each of the specified ListenIPs.

Long[] getListenIPTotalRequests(String[] listen_ip_addresses)

getListenIPs()

Gets the list of all IP addresses that Virtual Servers are listening on.

String[] getListenIPs()

getLocationLoad(names) throws InvalidInput, InvalidObjectName

The mean load metric for this location, for each of the named Locations.

Integer[] getLocationLoad(String[] names)

getLocationResponses(names) throws InvalidInput, InvalidObjectName

Number of A records that have been altered to point to this location, for each of the named Locations.

Integer[] getLocationResponses(String[] names)

getLocations()

Gets the list of Locations configured.

String[] getLocations()

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**
Current connections established to this node, includes idle connections, for each of the specified Nodes.

```java
Integer[] getNodeCurrentConn(
    System.Stats.Node[] nodes
)
```

**getNodeCurrentRequests( nodes ) throws InvalidInput, InvalidObjectName**
Active connections established to this node, does not include idle connections, for each of the specified Nodes.

```java
Integer[] getNodeCurrentRequests(
    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.

```java
Integer[] getNodeErrors(
    System.Stats.Node[] nodes
)
```

**getNodeFailures( nodes ) throws InvalidInput, InvalidObjectName**
Failures of this node, for each of the specified Nodes.

```java
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.

```java
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.

```java
Integer[] getNodeNewConn(
    System.Stats.Node[] nodes
)
```

**getNodeNumber()**
The number of nodes on this system (includes IPv4 and IPv6 nodes).

```java
Integer getNodeNumber()
```
System.Stats

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()
```
**Function Reference**

**System.Stats**

---

**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()
```

---

**getNumberSNMPGetBulkRequests()**
SNMP GetBulkRequests received.

```
Integer getNumberSNMPGetBulkRequests()
```

---

**getNumberSNMPGetNextRequests()**
SNMP GetNextRequests received.

```
Integer getNumberSNMPGetNextRequests()
```

---

**getNumberSNMPGetRequests()**
SNMP GetRequests received.

```
Integer getNumberSNMPGetRequests()
```

---

**getNumberSNMPUnauthorisedRequests()**
SNMP requests dropped due to access restrictions.

```
Integer getNumberSNMPUnauthorisedRequests()
```
getPerLocationServiceDraining( per_location_services ) throws InvalidInput, InvalidObjectName

The draining state of this location for this GLB Service, for each of the specified PerLocationServices.

```
System.Stats.PerLocationServiceDraining[] getPerLocationServiceDraining(
    System.Stats.PerLocationService[] per_location_services
)
```

getPerLocationServiceFrontendState( per_location_services ) throws InvalidInput, InvalidObjectName

The frontend state of this location for this GLB Service, for each of the specified PerLocationServices.

```
System.Stats.PerLocationServiceFrontendState[] getPerLocationServiceFrontendState(
    System.Stats.PerLocationService[] per_location_services
)
```

getPerLocationServiceLoad( per_location_services ) throws InvalidInput, InvalidObjectName

The load metric for this location for this GLB Service, for each of the specified PerLocationServices.

```
Integer[] getPerLocationServiceLoad(
    System.Stats.PerLocationService[] per_location_services
)
```

getPerLocationServiceMonitorState( per_location_services ) throws InvalidInput, InvalidObjectName

The monitor state of this location for this GLB Service, for each of the specified PerLocationServices.

```
System.Stats.PerLocationServiceMonitorState[] getPerLocationServiceMonitorState(
    System.Stats.PerLocationService[] per_location_services
)
```

getPerLocationServiceResponses( per_location_services ) throws InvalidInput, InvalidObjectName

Number of A records that have been altered to point to this location for this GLB Service, for each of the specified PerLocationServices.

```
Integer[] getPerLocationServiceResponses(
    System.Stats.PerLocationService[] per_location_services
)
```

getPerLocationServiceState( per_location_services ) throws InvalidInput, InvalidObjectName

The state of this location for this GLB Service, for each of the specified PerLocationServices.

```
System.Stats.PerLocationServiceState[] getPerLocationServiceState(
    System.Stats.PerLocationService[] per_location_services
)
```

getPerLocationServices()

Retrieves the list of available PerLocationServices.

```
System.Stats.PerLocationService[] getPerLocationServices()
```
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.
System.Stats

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

Current connections established to a node, includes idle connections, for each of the specified PerPoolNodes.

Integer[] getPerPoolNodeCurrentConn(
    System.Stats.PerPoolNode[] per_pool_nodes
)

getPerPoolNodeCurrentRequests( per_pool_nodes ) throws InvalidInput, InvalidObjectName

Active connections established to this node, does not include idle connections, for each of the specified PerPoolNodes.

Integer[] getPerPoolNodeCurrentRequests(
    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
)
Function Reference

System.Stats

getPerPoolNodeL4StatelessBuckets( per_pool_nodes ) throws InvalidInput, InvalidObjectName

Number of hash buckets occupied for this node for L4 stateless processing, for each of the specified PerPoolNodes.

```
Integer[] getPerPoolNodeL4StatelessBuckets(
    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()
```

getPerPoolNodePktsFromNode( per_pool_nodes ) throws InvalidInput, InvalidObjectName

Packets received from this node, for each of the specified PerPoolNodes.

```
Long[] getPerPoolNodePktsFromNode(
    System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodePktsToNode( per_pool_nodes ) throws InvalidInput, InvalidObjectName

Packets sent to this node, for each of the specified PerPoolNodes.

```
Long[] getPerPoolNodePktsToNode(
    System.Stats.PerPoolNode[] per_pool_nodes
)
```

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.

```java
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.

```java
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.

```java
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.

```java
Integer[] getPerPoolNodeTotalConn(
    System.Stats.PerPoolNode[] per_pool_nodes
)
```

getPerPoolNodes()

Retrieves the list of available PerPoolNodes.

```java
System.Stats.PerPoolNode[] getPerPoolNodes()
```

getPoolAlgorithm( names ) throws InvalidInput, InvalidObjectName

The load-balancing algorithm the pool uses, for each of the named Pools.

```java
System.Stats.PoolAlgorithm[] getPoolAlgorithm(
    String[] names
)
```

getPoolBwLimitBytesDrop( names ) throws InvalidInput, InvalidObjectName

Bytes dropped by this pool due to BW Limits, for each of the named Pools.

```java
Long[] getPoolBwLimitBytesDrop(
    String[] names
)
```
Function Reference

System.Stats

getPoolBwLimitPktsDrop( names ) throws InvalidInput, InvalidObjectName

Number of packets dropped by this pool due to BW Limits, for each of the named Pools.

Long[] getPoolBwLimitPktsDrop(
    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
)

getPoolConnsQueued( names )

Total connections currently queued to this pool, for each of the named Pools.

Integer[] getPoolConnsQueued(
    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
)

getPoolMaxQueueTime( names )

Maximum time a connection was queued for, over the last second, for each of the named Pools.

Integer[] getPoolMaxQueueTime(
    String[] names
)

getPoolMeanQueueTime( names )

Mean time a connection was queued for, over the last second, for each of the named Pools.

Integer[] getPoolMeanQueueTime(
    String[] names
)
Brocade Virtual Traffic Manager: Control API Guide

System.Stats Function Reference

getPoolMinQueueTime( names )
Minimum time a connection was queued for, over the last second, for each of the named Pools.

```
Integer[] getPoolMinQueueTime(
    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
)
```

gPoolNumber()
The number of pools on this system.

```
Integer getPoolNumber()
```

gPoolPersistence( names ) throws InvalidInput, InvalidObjectName
The session persistence method this pool uses, for each of the named Pools.

```
System.Stats.PoolPersistence[] getPoolPersistence(
    String[] names
)
```

gPoolQueueTimeouts( names )
Total connections that timed-out while queued, for each of the named Pools.

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

gPoolSessionMigrated( 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
)
```

gPoolState( names ) throws InvalidInput, InvalidObjectName
The state of this pool, for each of the named Pools.

```
System.Stats.PoolState[] getPoolState(
    String[] names
)
```

gPoolTotalConn( 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
)

getRuleExecutionTimeWarnings( names ) throws InvalidInput, InvalidObjectName
Number of times this TrafficScript rule has exceeded the execution time warning threshold, for each of the named Rules.

Integer[] getRuleExecutionTimeWarnings(
    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.

```java
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.

```java
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.

```java
Integer[] getRuleRetries(
    String[] names
)
```

**getRules()**

Gets the list of Rules configured.

```java
String[] getRules()
```

**getServiceLevelConforming(names) throws InvalidInput, InvalidObjectName**

Percentage of requests associated with this SLM class that are conforming, for each of the named ServiceLevels.

```java
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.

```java
Integer[] getServiceLevelCurrentConns(
    String[] names
)
```

**getServiceLevelIsOK(names) throws InvalidInput, InvalidObjectName**

Indicates if this SLM class is currently conforming, for each of the named ServiceLevels.

```java
System.Stats.ServiceLevelIsOK[] getServiceLevelIsOK(
    String[] names
)
```

**getServiceLevelNumber()**

The number of SLM classes defined.

```java
Integer getServiceLevelNumber()
```
**System.Stats Function Reference**

**getServiceLevelResponseMax( names ) throws InvalidInput, InvalidObjectName**

Maximum response time (ms) in the last second for this SLM class, for each of the named ServiceLevels.

```java
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.

```java
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.

```java
Integer[] getServiceLevelResponseMin(
    String[] names
)
```

**getServiceLevelTotalConn( names ) throws InvalidInput, InvalidObjectName**

Requests handled by this SLM class, for each of the named ServiceLevels.

```java
Integer[] getServiceLevelTotalConn(
    String[] names
)
```

**getServiceLevelTotalNonConf( names ) throws InvalidInput, InvalidObjectName**

Non-conforming requests handled by this SLM class, for each of the named ServiceLevels.

```java
Integer[] getServiceLevelTotalNonConf(
    String[] names
)
```

**getServiceLevels()**

Gets the list of Service Level Monitoring classes configured.

```java
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 ServicePros.

```java
Integer[] getServiceProtLastRefusalTime(
    String[] names
)
```

**getServiceProtNumber()**

The number of service protection classes defined.

```java
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.

```java
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.

```java
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.

```java
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.

```java
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.

```java
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.

```java
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.

```java
Integer[] getServiceProtRefusalSize(
    String[] names
)
**SystemServiceProtTotalRefusal(String names)** throws InvalidInput, InvalidObjectName
Connections refused by this service protection class, for each of the named ServiceProts.

```java
Integer[] getServiceProtTotalRefusal(String[] names)
```

**getServiceProts()**
Gets the list of Service Protection Classes configured.

```java
String[] getServiceProts()
```

**getSslCacheEntries()**
The total number of SSL sessions stored in the server cache.

```java
Integer getSslCacheEntries()
```

**getSslCacheEntriesMax()**
The maximum number of SSL entries in the server cache.

```java
Integer getSslCacheEntriesMax()
```

**getSslCacheHitRate()**
The percentage of SSL server cache lookups that succeeded.

```java
Integer getSslCacheHitRate()
```

**getSslCacheHits()**
Number of times a SSL entry has been successfully found in the server cache.

```java
Integer getSslCacheHits()
```

**getSslCacheLookups()**
Number of times a SSL entry has been looked up in the server cache.

```java
Integer getSslCacheLookups()
```

**getSslCacheMisses()**
Number of times a SSL entry has not been available in the server cache.

```java
Integer getSslCacheMisses()
```

**getSslCacheOldest()**
The age of the oldest SSL session in the server cache (in seconds).

```java
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()

**getSslCipherAESGCMDecrypts()**
Bytes decrypted with AES-GCM.
Integer getSslCipherAESGCMDecrypts()

**getSslCipherAESGCMEncrypts()**
Bytes encrypted with AES-GCM.
Integer getSslCipherAESGCMEncrypts()

**getSslCipherDESDecrypts()**
Bytes decrypted with DES.
Integer getSslCipherDESDecrypts()

**getSslCipherDESEncrypts()**
Bytes encrypted with DES.
Integer getSslCipherDESEncrypts()

**getSslCipherDHAgreements()**
Number of Diffie Hellman key agreements.
Integer getSslCipherDHAgreements()

**getSslCipherDHGenerates()**
Number of Diffie Hellman keys generated.
Integer getSslCipherDHGenerates()
getSslCipherDSASigns()
Number of DSA signing operations.
Integer getSslCipherDSASigns()

getSslCipherDSAVerifies()
Number of DSA verifications.
Integer getSslCipherDSAVerifies()

getSslCipherDecrypts()
Bytes decrypted with a symmetric cipher.
Integer getSslCipherDecrypts()

getSslCipherECDHAgreements()
Number of Elliptic Curve Diffie Hellman key agreements.
Integer getSslCipherECDHAgreements()

getSslCipherECDHGenerates()
Number of Elliptic Curve Diffie Hellman keys generated.
Integer getSslCipherECDHGenerates()

getSslCipherECDSASigns()
Number of ECDSA signing operations.
Integer getSslCipherECDSASigns()

getSslCipherECDSAVerifies()
Number of ECDSA verifications.
Integer getSslCipherECDSAVerifies()

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()**
Formerly provided the number of SSLv2 handshakes, now deprecated.

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()

getSslHandshakeTLSv12()
Number of TLSv1.2 handshakes.
Integer getSslHandshakeTLSv12()

getSslOcspStaplingCacheCount()
The number of entries in the OCSP stapling cache.
Integer getSslOcspStaplingCacheCount()

getSslOcspStaplingCount()
The number of outgoing OCSP requests for OCSP stapling.
Integer getSslOcspStaplingCount()

getSslOcspStaplingFailureCount()
The number of failed outgoing OCSP requests for OCSP stapling.
Integer getSslOcspStaplingFailureCount()

getSslOcspStaplingGoodCount()
The number of 'good' OCSP responses for OCSP stapling.
Integer getSslOcspStaplingGoodCount()

getSslOcspStaplingRevokedCount()
The number of 'revoked' OCSP responses for OCSP stapling.
Integer getSslOcspStaplingRevokedCount()

getSslOcspStaplingSuccessCount()
The number of successful outgoing OCSP requests for OCSP stapling.
Integer getSslOcspStaplingSuccessCount()
**getSslOcspStaplingUnknownCount()**
The number of 'unknown' OCSP requests for OCSP stapling.

```
Integer getSslOcspStaplingUnknownCount()
```

**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()
```

**getSteelheadNumber()**
The number of Steelheads.
```
Integer getSteelheadNumber()
```

**getSteelheadOptimized( names ) throws InvalidInput, InvalidObjectName**
The current number of connections being forwarded to the Cloud Steelhead for optimization, for each of the named Steelheads.
```
Integer[] getSteelheadOptimized(
    String[] names
)
```

**getSteelheads()**
Gets the list of Cloud Steelheads configured.
```
String[] getSteelheads()
```

**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.
Function Reference

System.Stats

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()

getTotalBackendServerErrors()
Total errors returned from the backend servers.
Integer getTotalBackendServerErrors()

getTotalBadDNSPackets()
Total number of malformed DNS response packets encountered from the backend servers.
Integer getTotalBadDNSPackets()

getTotalBytesIn()
Bytes received by the traffic manager from clients.
System.Stats Function Reference

Long getTotalBytesIn()

getTotalBytesOut()
Bytes sent by the traffic manager to clients.
Long getTotalBytesOut()

getTotalConn()
Total number of TCP connections received.
Integer getTotalConn()

getTotalCurrentConn()
Number of TCP connections currently established.
Integer getTotalCurrentConn()

getTotalDNSResponses()
Total number of DNS response packets handled.
Integer getTotalDNSResponses()

getTotalRequests()
Total number of TCP requests received.
Integer getTotalRequests()

getTotalTransactions()
Total number of TCP requests being processed, after applying TPS limits.
Integer getTotalTransactions()

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.

  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.
System.Stats

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 vTM software has been operational for (this value will wrap if it has been running for more than 497 days).
Integer getUpTime()

getUserCounter64Value( names ) throws InvalidInput, InvalidObjectName
The value of the 64-bit user counter, for each of the named UserCounter64s.

Long[] getUserCounter64Value(
    String[] names
)

getUserCounter64s()
Gets the list of 64-bit User counters configured.

String[] getUserCounter64s()

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()
```

getVirtualserverBwLimitBytesDrop( names ) throws InvalidInput, InvalidObjectName
Number of bytes dropped by this virtual server due to BW Limits, for each of the named Virtualservers.

```
Long[] getVirtualserverBwLimitBytesDrop{
    String[] names
}
```

getVirtualserverBwLimitPktsDrop( names ) throws InvalidInput, InvalidObjectName
Number of packets dropped by this virtual server due to BW Limits, for each of the named Virtualservers.

```
Long[] getVirtualserverBwLimitPktsDrop{
    String[] names
}
```

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
}
```

getVirtualserverCertStatusRequests( names )
Number of incoming TLS handshakes for this virtual server with certificate status requests, for each of the named Virtualservers.

```
Integer[] getVirtualserverCertStatusRequests{
    String[] names
}
```

getVirtualserverCertStatusResponses( names )
Number of incoming TLS handshakes for this virtual server to which certificate status responses were attached, for each of the named Virtualservers.
getVirtualserverCertStatusResponses(String[] names) throws InvalidInput

Number of certificate status responses received by this virtual server, for each of the named Virtualservers.

getVirtualserverConnectTimedOut(String[] names) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'connect_timeout' interval was exceeded, for each of the named Virtualservers.

getVirtualserverConnectionErrors(String[] names)

Number of transaction or protocol errors in this virtual server, for each of the named Virtualservers.

getVirtualserverConnectionFailures(String[] names)

Number of connection failures in this virtual server, for each of the named Virtualservers.

getVirtualserverCurrentConn(String[] names) throws InvalidInput, InvalidObjectName

TCP connections currently established to this virtual server, for each of the named Virtualservers.

getVirtualserverDataTimedOut(String[] names) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'timeout' interval was exceeded, for each of the named Virtualservers.

getVirtualserverDirectReplies(String[] names) throws InvalidInput, InvalidObjectName

Direct replies from this virtual server, without forwarding to a node, for each of the named Virtualservers.

getVirtualserverDiscard(String[] names) throws InvalidInput, InvalidObjectName

Connections discarded by this virtual server, for each of the named Virtualservers.
Function Reference

System.Stats

}  

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
)

getVirtualserverL4TCPConnectResets( names ) throws InvalidInput, InvalidObjectName

Number of TCP connections reset by this virtual server because the forward traffic cannot be processed, for each of the named Virtualservers.

Integer[] getVirtualserverL4TCPConnectResets(
   String[] names
)

getVirtualserverL4UDPUnreachables( names ) throws InvalidInput, InvalidObjectName

Number of ICMP error responses sent to the client by this virtual server because the forward traffic cannot be processed, for each of the named Virtualservers.

Integer[] getVirtualserverL4UDPUnreachables(
   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
)

getVirtualserverMaxDurationTimedOut( names ) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'max_transaction_duration' interval was exceeded, for each of the named Virtualservers.

Integer[] getVirtualserverMaxDurationTimedOut(
   String[] names
)
getVirtualserverNumber()

The number of virtual servers.

Integer getVirtualserverNumber()

getVirtualserverPktsIn( names ) throws InvalidInput, InvalidObjectName

Packets received by this virtual server from clients, for each of the named Virtualservers.

Long[] getVirtualserverPktsIn{
    String[] names
}

getVirtualserverPktsOut( names ) throws InvalidInput, InvalidObjectName

Packets sent by this virtual server to clients, for each of the named Virtualservers.

Long[] getVirtualserverPktsOut{
    String[] names
}

getVirtualserverPort( names ) throws InvalidInput, InvalidObjectName

The port the virtual server listens on, for each of the named Virtualservers.

Integer[] getVirtualserverPort{
    String[] names
}

getVirtualserverProcessingTimedOut( names ) throws InvalidInput, InvalidObjectName

Connections closed by this virtual server because the 'timeout' interval was exceeded while waiting for rules or external processing, for each of the named Virtualservers.

Integer[] getVirtualserverProcessingTimedOut{
    String[] names
}

getVirtualserverProtocol( names ) throws InvalidInput, InvalidObjectName

The protocol the virtual server is operating, for each of the named Virtualservers.

    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.

```java
Integer[] getVirtualserverSIPTotalCalls(
    String[] names
)
```

getVirtualserverTotalConn( names ) throws InvalidInput, InvalidObjectName

Formerly provided the number of requests received by this virtual server, now deprecated, for each of the named Virtualservers.

```java
Integer[] getVirtualserverTotalConn(
    String[] names
)
```

getVirtualserverTotalDgram( names ) throws InvalidInput, InvalidObjectName

UDP datagrams processed by this virtual server, for each of the named Virtualservers.

```java
Integer[] getVirtualserverTotalDgram(
    String[] names
)
```

getVirtualserverTotalHTTP1Requests( names ) throws InvalidInput, InvalidObjectName

HTTP/1.x Requests received by this virtual server, for each of the named Virtualservers.

```java
Long[] getVirtualserverTotalHTTP1Requests(
    String[] names
)
```

getVirtualserverTotalHTTP2Requests( names ) throws InvalidInput, InvalidObjectName

HTTP/2 Requests received by this virtual server, for each of the named Virtualservers.

```java
Long[] getVirtualserverTotalHTTP2Requests(
    String[] names
)
```

getVirtualserverTotalHTTPRequests( names ) throws InvalidInput, InvalidObjectName

HTTP Requests received by this virtual server, for each of the named Virtualservers.

```java
Long[] getVirtualserverTotalHTTPRequests(
    String[] names
)
```

getVirtualserverTotalRequests( names ) throws InvalidInput, InvalidObjectName

Requests received by this virtual server, for each of the named Virtualservers.

```java
Long[] getVirtualserverTotalRequests(
    String[] names
)
```
**getVirtualserverUdpTimedOut**

Connections closed by this virtual server because the 'udp_timeout' interval was exceeded, for each of the named Virtual servers.

```java
Integer[] getVirtualserverUdpTimedOut(String[] names)
```

**getVirtualservers()**

Gets the list of Virtual Servers configured and enabled.

```java
String[] getVirtualservers()
```

**getWebCacheEntries()**

The number of items in the web cache.

```java
Integer getWebCacheEntries()
```

**getWebCacheHitRate()**

The percentage of web cache lookups that succeeded.

```java
Integer getWebCacheHitRate()
```

**getWebCacheHits()**

Number of times a page has been successfully found in the web cache.

```java
Long getWebCacheHits()
```

**getWebCacheLookups()**

Number of times a page has been looked up in the web cache.

```java
Long getWebCacheLookups()
```

**getWebCacheMaxEntries()**

The maximum number of items in the web cache.

```java
Integer getWebCacheMaxEntries()
```

**getWebCacheMemMaximum()**

The maximum amount of memory the web cache can use in kilobytes.

```java
Integer getWebCacheMemMaximum()
```

**getWebCacheMemUsed()**

Total memory used by the web cache in kilobytes.

```java
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()

getWebCacheURLStoreAllocated()
Amount of allocated space in the web cache URL store.
Long getWebCacheURLStoreAllocated()

getWebCacheURLStoreFree()
Amount of free space in the web cache URL store.
Long getWebCacheURLStoreFree()

getWebCacheURLStoreSize()
Total amount of space in the web cache URL store.
Long getWebCacheURLStoreSize()

getWebCacheURLStoreTotalAllocations()
Total number of allocations for the web cache URL store.
Long getWebCacheURLStoreTotalAllocations()

getWebCacheURLStoreTotalFailures()
Total number of allocation failures for the web cache URL store.
Long getWebCacheURLStoreTotalFailures()

getWebCacheURLStoreTotalFrees()
Total number of blocks freed in the web cache URL store.
Long getWebCacheURLStoreTotalFrees()

getZxtmNumber()
The number of traffic managers in the cluster.
Integer getZxtmNumber()

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.PerLocationService

Represents a PerLocationService object.

struct System.Stats.PerLocationService {
    # The name of the location.
    String LocationName;

    # The name of the GLB Service.
    String Name;
}

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;
}

Enumerations

System.Stats.NodeState

enum System.Stats.NodeState {
    alive,
    dead,
    unknown
}
System.Stats.PerLocationServiceDraining
eum System.Stats.PerLocationServiceDraining {
    draining,
    active
}

System.Stats.PerLocationServiceFrontendState
eum System.Stats.PerLocationServiceFrontendState {
    alive,
    dead
}

System.Stats.PerLocationServiceMonitorState
eum System.Stats.PerLocationServiceMonitorState {
    alive,
    dead
}

System.Stats.PerLocationServiceState
eum System.Stats.PerLocationServiceState {
    alive,
    dead
}

System.Stats.PerPoolNodeState
eum System.Stats.PerPoolNodeState {
    alive,
    dead,
    unknown,
    draining,
    drainingtodelete
}

System.Stats.PoolAlgorithm
eum System.Stats.PoolAlgorithm {
    roundrobin,
    weightedRoundRobin,
    perceptive,
    leastConnections,
    fastestResponseTime,
    random,
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,
udpsstreaming,
udp,
dns,
genericserverfirst,
genericclientfirst,
dnstcp,
sipudp,
siptcp,
rtsp,
stream,
14acceltcp,
14acceludp,
14accelgeneric,
14accelstateless

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.
Methods

rebootSystem()
Perform a system reboot.

void rebootSystem()

regenerateUUID() throws DeploymentError
Regenerate the UUID of this traffic manager.

String regenerateUUID()

restartAFM() throws InvalidOperation, LicenseError
Restart the Brocade Virtual Web Application Firewall on all machines. Any connections currently using Brocade vWAF 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()

AFM

URI: http://soap.zeus.com/zxtm/1.0/AFM/
The AFM interface allows management of the Brocade Virtual Web Application Firewall.

Methods

disable() throws InvalidOperation, LicenseError
Disables the Brocade Virtual Web Application Firewall on the traffic manager.

void disable()
**enable() throws InvalidOperation, LicenseError**
Enables the Brocade Virtual Web Application Firewall on the traffic manager.

```java
void enable()
```

**getAdminInternalRESTPort() throws InvalidOperation**
Get the Application Firewall Internal REST API port, this port should not be accessed directly.

```java
Integer getAdminInternalRESTPort()
```

**getAdminMasterPort() throws InvalidOperation**
Get the Application Firewall XML Master port, this port is used on all IP addresses.

```java
Integer getAdminMasterPort()
```

**getAdminServerPort() throws InvalidOperation**
Get the Application Firewall Administration Server port, this port is open only on localhost.

```java
Integer getAdminServerPort()
```

**getAdminSlavePort() throws InvalidOperation**
Get the Application Firewall XML Slave port, this port is used on all IP addresses.

```java
Integer getAdminSlavePort()
```

**getClusterState()**
Get state data for the Brocade Virtual Web Application Firewall across all machines in the cluster.

```java
AFM.State[] getClusterState()
```

**getDeciderBasePort() throws InvalidOperation**
Get the port to which the Enforcer rule should send traffic so it can be distributed between the decider processes.

```java
Unsigned Integer getDeciderBasePort()
```

**getDeciderServerBasePort() throws InvalidOperation**
Get the base port from which the Application Firewall decider processes should run. Ports will be used sequentially above this for each additional decider process that runs.

```java
Integer getDeciderServerBasePort()
```

**getInternalDeciderBasePort() throws InvalidOperation**
Get the Application Firewall internal decider communication base port. The Application Firewall will use ports sequentially above this for internal communication. These ports are bound only to localhost.

```java
Integer getInternalDeciderBasePort()
```

**getNumberOfDeciders() throws InvalidOperation**
Get the number of decider processes
Function Reference

Integer getNumberOfDeciders()

**getUpdateExternControlCenterPort() throws InvalidOperation**
Get the Application Firewall Updater External Control Center port, this port is used on all IP addresses.
Integer getUpdateExternControlCenterPort()

**getUpdateGUIBackendPort() throws InvalidOperation**
Get the Application Firewall Updater GUI Backend port, this port is used on all IP addresses.
Integer getUpdateGUIBackendPort()

**getUpdateGUIServerPort() throws InvalidOperation**
Get the Application Firewall Updater GUI Server port, this port is used on all IP addresses.
Integer getUpdateGUIServerPort()

**getUpdaterPort() throws InvalidOperation**
Get the Application Firewall Updater Slave port, this port is used on all IP addresses.
Unsigned Integer getUpdaterPort()

**getVersion()**
Get the version of the Brocade Virtual Web Application Firewall installed on the traffic manager. Returns an empty string if Brocade vWAF is not installed.
String getVersion()

**setAdminInternalRESTPort( port ) throws InvalidOperation, InvalidInput**
Set the Application Firewall Internal REST API port, this port should not be accessed directly.
void setAdminInternalRESTPort(
    Integer port
)

**setAdminMasterPort( port ) throws InvalidOperation, InvalidInput**
Set the Application Firewall XML Master port, this port is used on all IP addresses.
void setAdminMasterPort(
    Integer port
)

**setAdminServerPort( port ) throws InvalidOperation, InvalidInput**
Set the Application Firewall Administration Server port, this port is open only on localhost.
void setAdminServerPort(
    Integer port
)

**setAdminSlavePort( port ) throws InvalidOperation, InvalidInput**
Set the Application Firewall XML Slave port, this port is used on all IP addresses.
void setAdminSlavePort(
    Integer port
)

setDeciderBasePort( value ) throws InvalidOperation, InvalidInput
Set the port to which the Enforcer rule should send traffic so it can be distributed between the decider processes.
void setDeciderBasePort{
    Unsigned Integer value
}

setDeciderServerBasePort( port ) throws InvalidOperation, InvalidInput
Set the base port from which the Application Firewall decider processes should run. Ports will be used sequentially above this for each additional decider process that runs.
void setDeciderServerBasePort{
    Integer port
}

setInternalDeciderBasePort( port ) throws InvalidOperation, InvalidInput
Set the Application Firewall internal decider communication base port. The Application Firewall will use ports sequentially above this for internal communication. These ports are bound only to localhost.
void setInternalDeciderBasePort{
    Integer port
}

setNumberOfDeciders( deciders ) throws InvalidOperation, InvalidInput
Set the number of decider processes
void setNumberOfDeciders{
    Integer deciders
}

setUpdateExternControlCenterPort( port ) throws InvalidOperation, InvalidInput
Set the Application Firewall Updater External Control Center port, this port is used on all IP addresses.
void setUpdateExternControlCenterPort{
    Integer port
}

setUpdateGUIBackendPort( port ) throws InvalidOperation, InvalidInput
Set the Application Firewall Updater GUI Backend port, this port is used on all IP addresses.
void setUpdateGUIBackendPort{
    Integer port
}

setUpdateGUIServerPort( port ) throws InvalidOperation, InvalidInput
Set the Application Firewall Updater GUI Server port, this port is used on all IP addresses.
void setUpdateGUIServerPort{
    Integer port
}
setUpdaterPort( value ) throws InvalidOperation, InvalidInput

Set the Application Firewall Updater Slave port, this port is used on all IP addresses.

```java
void setUpdaterPort(
    Unsigned Integer value
)
```

uninstall() throws InvalidOperation

Uninstalls the Brocade Virtual Web Application Firewall on the traffic manager.

```java
void uninstall()
```

**Structures**

**AFM.BasicStatus**

Contains basic Brocade Virtual Web Application Firewall runtime status information.

```java
struct AFM.BasicStatus {
    # Whether or not Brocade vWAF is installed.
    String installed;

    # Whether or not Brocade vWAF is running.
    String running;

    # The version of Brocade vWAF installed.
    String version;

    # Whether or not the machine is clustered with the local Brocade vWAF.
    String clustered;
}
```

**AFM.ClusterStatus**

Contains a Brocade Virtual Web Application Firewall state message.

```java
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 Brocade Virtual Web Application Firewall installation.

```java
struct AFM.State {
    # Name of the machine this information is from.
    String machine;

    # Describes the basic runtime status of Brocade vWAF on a machine.
    AFM.BasicStatus basicstatus;

    # State messages from the Brocade vWAF on the machine.
    AFM.StateMessage[] messages;
}
Location

URI: http://soap.zeus.com/zxtm/1.0/Location/

The Location interface allows management of traffic manager locations. Using this interface, you can create, delete and rename Locations, and manage their configuration.

Methods

addLocation(locations, info) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, InvalidInput, DeploymentError, LicenseError, InvalidOperation

Adds locations. Configuration for the new locations will be based on the specified locations

```java
void addLocation(
    String[] locations
    Location.TypeInfo[] info
)
```

deleteLocation(locations) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, LicenseError

Delete the named Location.

```java
void deleteLocation{
    String[] locations
}
```

disable(location) throws InvalidInput, ObjectDoesNotExist, LicenseError

Disable support for configuration locations, setting all configuration values to those for the specified location.

```java
void disable{
    String location
}
```
**enable() throws LicenseError, InvalidOperation**

Enable support for configuration locations.

```java
void enable()
```

**getCoordinates(locations) throws ObjectDoesNotExist, LicenseError, InvalidOperation**

Get the coordinates for the named locations.

```java
Location.Coordinates[] getCoordinates(
    String[] locations
)
```

**getLocations()**

Get the names of all the configured locations.

```java
String[] getLocations()
```

**getNote(locations) throws ObjectDoesNotExist, LicenseError, InvalidOperation**

Get the note for each of the named locations

```java
String[] getNote(
    String[] locations
)
```

**getTrafficManagerLocation(traffic_managers) throws ObjectDoesNotExist, LicenseError**

Gets the location that the named traffic managers are in.

```java
String[] getTrafficManagerLocation(
    String[] traffic_managers
)
```

**getType(locations) throws ObjectDoesNotExist, LicenseError, InvalidOperation**

Gets a location’s type, either config or glb. GLB locations contain no traffic managers, and are only used for global load balancing.

```java
String[] getType(
    String[] locations
)
```

**renameLocation(locations, new_names) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName, DeploymentError, LicenseError**

Rename the named Locations.

```java
void renameLocation(
    String[] locations
    String[] new_names
)
```
setCoordinates( locations, coordinates ) throws ObjectDoesNotExist, DeploymentError, LicenseError, InvalidOperation

Set the coordinates for the named locations. Coordinates are only needed for global load balancing.

```java
void setCoordinates(
    String[] locations
    Location.Coordinates[] coordinates
)
```

setNote( locations, values ) throws ObjectDoesNotExist, DeploymentError, LicenseError, InvalidOperation

Set the note for each of the named locations

```java
void setNote(
    String[] locations
    String[] values
)
```

setTrafficManagerLocation( traffic_managers, locations ) throws ObjectDoesNotExist, LicenseError, DeploymentError, InvalidOperation

Sets the location that the named traffic managers are in.

```java
void setTrafficManagerLocation(
    String[] traffic_managers
    String[] locations
)
```

setType( locations, type_info ) throws ObjectInUse, ObjectDoesNotExist, LicenseError, InvalidInput, InvalidOperation

Sets a location’s type, either config or glb. GLB locations contain no traffic managers, and are only used for global load balancing.

```java
void setType(
    String[] locations
    Location.TypeInfo[] type_info
)
```

### Structures

**Location.Coordinates**

This structure contains the co-ordinates for a location.

```java
struct Location.Coordinates {
    # The longitude of the location.
    Double longitude;

    # The latitude of the location.
    Double latitude;
}
```

**Location.TypeInfo**

This structure contains information required when adding a location.
Users

URI: http://soap.zeus.com/zxtm/1.0/Users/

The Users interface allows management of users of Brocade Virtual Traffic Manager. Using this interface, you can create and delete users, assign them to permission groups and manage their configuration.

Methods

**addUser( user, password, group ) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidInput**

Add a new local user to Brocade Virtual Traffic Manager.

```java
void addUser(
    String user
    String password
    String group
)
```

**changePassword( user, newPassword ) throws ObjectDoesNotExist, InvalidInput**

Change password for a given user.

```java
void changePassword(
    String user
    String newPassword
)
```

**deleteUser( user ) throws ObjectDoesNotExist, DeploymentError**

Delete a local user from Brocade Virtual Traffic Manager.

```java
void deleteUser(
    String user
)
```

**listGroups()**

List all groups of Brocade Virtual Traffic Manager.

```java
String[] listGroups()
```

**listUsers()**

List all users of Brocade Virtual Traffic Manager.
GLB.Service

URI: http://soap.zeus.com/zxtm/1.0/GLB/Service/

The GLB.Service interface allows management of Global Load Balancing Services. Using this interface, you can create, delete and rename pool objects, and manage their configuration.

Methods

addDNSSECMapping( names, mappings ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add a set of DNSSEC domain to key mappings to the GLB services specified.

```java
void addDNSSECMapping(
    String[] names
    GLB.Service.DNSSECMapping[][] mappings
)
```

addDomains( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

For each named GLB service, add new DNS domain names to the list of domains to load balance.

```java
void addDomains(
    String[] names
    String[][] values
)
```

addDraining( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add the list of locations that are draining for this service.

```java
void addDraining(
    String[] names
    String[][] values
)
```

addGLBService( names, domains ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidInput, LicenseError

Add each of the named GLB Services, using the domain lists for each.

```java
void addGLBService(
    String[] names
    String[][] domains
)
```
addLastResortResponse( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add Last Resort Response.

```java
void addLastResortResponse(
    String[] names
    String[][] values
)
```

addLocalIPAddresses( names, localips ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

This method is now deprecated and is replaced by addServiceIPAddresses.

```java
void addLocalIPAddresses(
    String[] names
    GLB.Service.LocalIPList[][] localips
)
```

addLocations( names, locations ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add new locations to each of the named GLB services.

```java
void addLocations(
    String[] names
    String[][] locations
)
```

addMonitors( names, monitors ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add Monitors to the named GLB Services.

```java
void addMonitors(
    String[] names
    GLB.Service.MonitorList[][] monitors
)
```

addMonitorsByLocation( location, names, monitors ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add Monitors to the named GLB Services. This is a location specific function, any action will operate on the specified location.

```java
void addMonitorsByLocation(
    String location
    String[] names
    GLB.Service.MonitorList[][] monitors
)
```

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

Add new rules to be run on DNS packets for each of the named GLB services. 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.

```java
void addRules(
```
addRulesByLocation(location, names, rules) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add new rules to be run on DNS packets for each of the named GLB services. 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. This is a location specific function, any action will operate on the specified location.

```java
void addRulesByLocation(
    String location
    String[] names
    GLB.Service.Rule[][] rules
)
```

addServiceIPAddresses(names, localips) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add Service IP addresses to the named GLB Services

```java
void addServiceIPAddresses(
    String[] names
    GLB.Service.LocalIPList[][] localips
)
```

deleteGLBService(names) throws ObjectDoesNotExist, DeploymentError

Delete each of the named GLB Services.

```java
void deleteGLBService(
    String[] names
)
```

getAlgorithm(names) throws ObjectDoesNotExist, LicenseError

Get the load balancing algorithm to use.

```java
GLB.Service.Algorithm[] getAlgorithm(
    String[] names
)
```

getAllMonitorsNeeded(names) throws ObjectDoesNotExist, LicenseError

Get whether all monitors are required to be working in a location for this service to be alive.

```java
Boolean[] getAllMonitorsNeeded(
    String[] names
)
```

getAutoFailback(names) throws ObjectDoesNotExist, LicenseError

Get whether automatic failback mode is enabled.

```java
Boolean[] getAutoFailback(
    String[] names
)
```
getAutoRecovery( names ) throws ObjectDoesNotExist, LicenseError
Get whether automatic recovery of the last location to fail is enabled.

Boolean[] getAutoRecovery(
    String[] names
)

getDNSSECMapping( names ) throws ObjectDoesNotExist, LicenseError
Get the load for the named GLB Services

GLB.Service.DNSSECMapping[][] getDNSSECMapping(
    String[] names
)

getAddressOnFailure( names ) throws ObjectDoesNotExist, LicenseError
Get whether "Disable on Failure" mode is enabled.

Boolean[] getDisableOnFailure(
    String[] names
)

getDomains( names ) throws ObjectDoesNotExist, LicenseError
Get the list of domain names to load balance, for each of the named GLB services

String[][] getDomains(
    String[] names
)

getDraining( names ) throws ObjectDoesNotExist, LicenseError
Get the list of locations that are draining for this service.

String[][] getDraining(
    String[] names
)

getEnabled( names ) throws ObjectDoesNotExist, LicenseError
Get whether we perform DNS manipulation.

Boolean[] getEnabled(
    String[] names
)

getGLBServiceNames()
Get the names of all of the configured GLB Services.

String[] getGLBServiceNames()

getGeoEffect( names ) throws ObjectDoesNotExist, LicenseError
Get the influence of locality on location choice

Unsigned Integer[] getGeoEffect(
    String[] names
)
**getLastResortResponse( names ) throws ObjectDoesNotExist, LicenseError**

Get Last Resort Response.

```java
String[][] getLastResortResponse(
    String[] names
)
```

**getLoad( names ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Get the load for the named GLB Services

```java
GLB.Service.Load[][] getLoad(
    String[] names
)
```

**getLocalIPAddresses( names ) throws ObjectDoesNotExist, LicenseError**

This method is now deprecated and is replaced by getServiceIPAddresses

```java
GLB.Service.LocalIPList[][] getLocalIPAddresses(
    String[] names
)
```

**getLocations( names ) throws ObjectDoesNotExist, LicenseError**

Get the locations configured for the named GLB services.

```java
String[][] getLocations(
    String[] names
)
```

**getLogEnabled( names ) throws ObjectDoesNotExist, LicenseError**

Get whether each of the named GLB services should log each connection.

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

**getLogEnabledByLocation( location, names ) throws ObjectDoesNotExist, LicenseError**

Get whether each of the named GLB services should log each connection. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getLogEnabledByLocation(
    String location
    String[] names
)
```

**getLogFilename( names ) throws ObjectDoesNotExist, LicenseError**

Get the name of the file used to store query logs, for each of the named GLB services.

```java
String[] getLogFilename(
    String[] names
)
```
Function Reference

**GLB.Service**

---

**getLogFilenameByLocation( location, names ) throws ObjectDoesNotExist, LicenseError**

Get the name of the file used to store query logs, for each of the named GLB services. This is a location specific function, any action will operate on the specified location.

```java
String[] getLogFilenameByLocation(
    String location
    String[] names
)
```

**getLogFormat( names ) throws ObjectDoesNotExist, LicenseError**

Get the log file format for each of the named GLB services.

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

**getLogFormatByLocation( location, names ) throws ObjectDoesNotExist, LicenseError**

Get the log file format for each of the named GLB services. This is a location specific function, any action will operate on the specified location.

```java
String[] getLogFormatByLocation(
    String location
    String[] names
)
```

**getMonitors( names ) throws ObjectDoesNotExist, LicenseError**

Get the Monitors configured for the named GLB Services

```java
GLB.Service.MonitorList[][] getMonitors(
    String[] names
)
```

**getMonitorsByLocation( location, names ) throws ObjectDoesNotExist, LicenseError**

Get the Monitors configured for the named GLB Services. This is a location specific function, any action will operate on the specified location.

```java
GLB.Service.MonitorList[][] getMonitorsByLocation(
    String location
    String[] names
)
```

**getReturnIPsOnFail( names ) throws ObjectDoesNotExist, LicenseError**

Get whether to return all or no IP addresses on a complete failure

```java
Boolean[] getReturnIPsOnFail(
    String[] names
)
```

**getRules( names ) throws ObjectDoesNotExist, LicenseError**

Get the rules that are run on DNS packets for each of the named GLB services.
GLB.Service.Rule[][] getRules{
    String[] names
}

getRulesByLocation( location, names ) throws ObjectDoesNotExist, LicenseError
Get the rules that are run on DNS packets for each of the named GLB services. This is a location specific function, any action will operate on the specified location.

GLB.Service.Rule[][] getRulesByLocation{
    String location
    String[] names
}

getServiceIPAddresses( names ) throws ObjectDoesNotExist, LicenseError
Get the Service IP addresses configured for the named GLB Services

GLB.Service.LocalIPList[][] getServiceIPAddresses{
    String[] names
}

getTTL( names ) throws ObjectDoesNotExist, LicenseError
Get the TTL for the DNS resource records handled by the GLB service.

Integer[] getTTL{
    String[] names
}

removeDNSSECMapping( names, mappings ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Remove the specified DNSSEC domain to key mappings.

void removeDNSSECMapping{
    String[] names
    GLB.Service.DNSSECMapping[][] mappings
}

removeDomains( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
For each named GLB service, remove DNS domain names from the list of domains to load balance

void removeDomains{
    String[] names
    String[][] values
}

removeDraining( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Remove the list of locations that are draining for this service.

void removeDraining{
    String[] names
    String[][] values
}
Function Reference

**removeLastResortResponse** *(names, values)* throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Remove Last Resort Response.

```java
void removeLastResortResponse(
    String[] names
    String[][] values
)
```

**removeLocalIPAddresses** *(names, localips)* throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

This method is now deprecated and is replaced by removeServiceIPAddresses

```java
void removeLocalIPAddresses(
    String[] names
    GLB.Service.LocalIPList[][] localips
)
```

**removeLocations** *(names, locations)* throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

For each of the named GLB services, remove locations.

```java
void removeLocations(
    String[] names
    String[][] locations
)
```

**removeMonitors** *(names, monitors)* throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Remove Monitors from the named GLB Services

```java
void removeMonitors(
    String[] names
    GLB.Service.MonitorList[][] monitors
)
```

**removeMonitorsByLocation** *(location, names, monitors)* throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Remove Monitors from the named GLB Services This is a location specific function, any action will operate on the specified location.

```java
void removeMonitorsByLocation(
    String location
    String[] names
    GLB.Service.MonitorList[][] monitors
)
```

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

For each of the named GLB services, remove rules from the list of rules that are run on DNS packets.

```java
void removeRules(
    String[] names
    String[][] rules
)
removeRulesByLocation( location, names, rules ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

For each of the named GLB services, remove rules from the list of rules that are run on DNS packets. This is a location specific function, any action will operate on the specified location.

```java
void removeRulesByLocation(
    String location
    String[] names
    String[][] rules
)
```

removeServiceIPAddresses( names, localips ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Remove Service IP addresses from the named GLB Services.

```java
void removeServiceIPAddresses(
    String[] names
    GLB.Service.LocalIPList[][] localips
)
```

renameGLBService( names, new_names ) throws ObjectDoesNotExist, InvalidInput, ObjectAlreadyExists, DeploymentError, LicenseError

Rename each of the named GLB Services.

```java
void renameGLBService(
    String[] names
    String[] new_names
)
```

setAlgorithm( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the load balancing algorithm to use.

```java
void setAlgorithm(
    String[] names
    GLB.Service.Algorithm[] values
)
```

setAllMonitorsNeeded( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set whether all monitors are required to be working in a location for this service to be alive.

```java
void setAllMonitorsNeeded(
    String[] names
    Boolean[] values
)
```

setAutoFailback( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set whether automatic failback mode is enabled.
void setAutoFailback(
    String[] names
    Boolean[] values
)

setAutoRecovery( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set whether automatic recovery of the last location to fail is enabled.

void setAutoRecovery(
    String[] names
    Boolean[] values
)

setDNSSECMappings( names, mappings ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the DNSSEC domain to key mappings to the GLB services specified. All previous mappings for this service will be removed.

void setDNSSECMappings(
    String[] names
    GLB.Service.DNSSECMapping[][][ mappings
)

setDisableOnFailure( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set whether "Disable on Failure" mode is enabled.

void setDisableOnFailure(
    String[] names
    Boolean[] values
)

setDomains( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the list of domain names to load balance, for each of the named GLB services

void setDomains(
    String[] names
    String[][][] values
)

setDraining( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the list of locations that are draining for this service.

void setDraining(
    String[] names
    String[][][] values
)
setEnabled( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set whether we perform DNS manipulation.

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

setGeoEffect( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the influence of locality on location choice

```java
void setGeoEffect(
    String[] names
    Unsigned Integer[] values
)
```

setLastResortResponse( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set Last Resort Response.

```java
void setLastResortResponse(
    String[] names
    String[][] values
)
```

setLoad( names, loads ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the load for the named GLB Services

```java
void setLoad(
    String[] names
    GLB.Service.Load[][] loads
)
```

setLocalIPAddresses( names, localips ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

This method is now deprecated and is replaced by setServiceIPAddresses

```java
void setLocalIPAddresses(
    String[] names
    GLB.Service.LocalIPList[][] localips
)
```

setLocations( names, locations ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the locations configured for each of the named GLB services.

```java
void setLocations(
    String[] names
    String[][][] locations
)
```
Function Reference

GLB.Service

Set whether each of the named GLB services should log each connection.

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

Set whether each of the named GLB services should log each connection. This is a location specific function, any action will operate on the specified location.

```java
void setLogEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```

Set the name of the file used to store query logs, for each of the named GLB services.

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

Set the name of the file used to store query logs, for each of the named GLB services. This is a location specific function, any action will operate on the specified location.

```java
void setLogFilenameByLocation(
    String location
    String[] names
    String[] values
)
```

Set the log file format for each of the named GLB services.

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

Set the log file format for each of the named GLB services. This is a location specific function, any action will operate on the specified location.

```java
void setLogFormatByLocation(
    String location
    String[] names
    String[] values
)
```
void setLogFormatByLocation(
    String location
    String[] names
    String[] values
)

setMonitors( names, monitors ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the Monitors configured for the named GLB Services

void setMonitors(
    String[] names
    GLB.Service.MonitorList[][] monitors
)

setMonitorsByLocation( location, names, monitors ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the Monitors configured for the named GLB Services. This is a location specific function, any action will operate on the specified location.

void setMonitorsByLocation(
    String location
    String[] names
    GLB.Service.MonitorList[][] monitors
)

setReturnIPsOnFail( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set whether to return all or no IP addresses on a complete failure

void setReturnIPsOnFail(
    String[] names
    Boolean[] values
)

setRules( names, rules ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the rules that are run on DNS packets for each of the named GLB services.

void setRules(
    String[] names
    GLB.Service.Rule[][] rules
)

setRulesByLocation( location, names, rules ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError
Set the rules that are run on DNS packets for each of the named GLB services. This is a location specific function, any action will operate on the specified location.

void setRulesByLocation(
    String location
    String[] names
    GLB.Service.Rule[][] rules
)
**setServiceIPAddresses(names, localips)** throws **ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Set the Service IP addresses configured for the named GLB Services

```java
void setServiceIPAddresses(
    String[] names
    GLB.Service.LocalIPList[][] localips
)
```

**setTTL(names, values)** throws **ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError**

Set The TTL for the DNS resource records handled by the GLB service.

```java
void setTTL(
    String[] names
    Integer[] values
)
```

**Structures**

**GLB.Service.DNSSECMapping**

This contains a mapping between DNS domains and DNSSEC keys used to alter signed responses.

```java
struct GLB.Service.DNSSECMapping {
    # The domain of these keys sign.
    String domain;

    # An array of DNSSEC key names.
    String[] keys;
}
```

**GLB.Service.Load**

This structure contains the load for a GLB location.

```java
struct GLB.Service.Load {
    # The glb location.
    String location;

    # The load metric at the corresponding location.
    Integer load;
}
```

**GLB.Service.LocalIPList**

This structure contains the list of IP addresses for a GLB location.

```java
struct GLB.Service.LocalIPList {
    # The glb location.
    String location;

    # The IP Addresses or IP Masks that are present at the corresponding
    # location.
    String[] addresses;
}
```
**GLB.Service.MonitorList**

This structure contains the list of monitors for a GLB location.

```java
struct GLB.Service.MonitorList {
    # The glb location.
    String location;
    # The monitors determining the health of the corresponding location.
    String[] monitors;
}
```

**GLB.Service.Rule**

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

```java
struct GLB.Service.Rule {
    # The name of the rule.
    String name;
    # Whether the rule is enabled or not.
    Boolean enabled;
}
```

**Enumerations**

**GLB.Service.Algorithm**

```java
enum GLB.Service.Algorithm {
    # Load
    load,
    # Geographic
    geo,
    # Adaptive
    hybrid,
    # Round Robin
    roundrobin,
    # Weighted Random
    weightedrandom,
    # Primary/Backup
    chained
}
```

**System.CloudCredentials**

URI: http://soap.zeus.com/zxtm/1.0/System/CloudCredentials/

The System.CloudCredentials interface allows management of Cloud Credentials. Using this interface, you can create, delete and rename sets of cloud credentials, and manage their configuration.
Methods

addCloudCredentials( class_names, class_values ) throws ObjectAlreadyExists, InvalidInput
Add new sets of cloud credentials.
void addCloudCredentials{
    String[] class_names
    System.CloudCredentials.CredentialsData[] class_values
}

copyCloudCredentials( class_names, new_names ) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError
Copy the named set of cloud credentials.
void copyCloudCredentials{
    String[] class_names
    String[] new_names
}

deleteCloudCredentials( class_names ) throws ObjectDoesNotExist, DeploymentError
Delete the named sets of cloud credentials.
void deleteCloudCredentials{
    String[] class_names
}

getApiServer( class_names ) throws ObjectDoesNotExist
Get the vcenter server hostname or IP address.
String[] getApiServer{
    String[] class_names
}

getApiServerByLocation( location, class_names ) throws ObjectDoesNotExist
Get the vcenter server hostname or IP address. This is a location specific function, any action will operate on the specified location.
String[] getApiServerByLocation{
    String location
    String[] class_names
}

getChangeProcessTimeout( class_names ) throws ObjectDoesNotExist
Get the amount of time change calls are allowed to take
Unsigned Integer[] getChangeProcessTimeout{
    String[] class_names
}
getChangeProcessTimeoutByLocation( location, class_names ) throws ObjectDoesNotExist

Get the amount of time change calls are allowed to take. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getChangeProcessTimeoutByLocation(
    String location
    String[] class_names
)

getCloudCredentialsNames()

Get the names of all the configured cloud credentials.

String[] getCloudCredentialsNames()

getCred1( class_names ) throws ObjectDoesNotExist

Get the cloud user name

String[] getCred1(
    String[] class_names
)

g getCred1ByLocation( location, class_names ) throws ObjectDoesNotExist

Get the cloud user name. This is a location specific function, any action will operate on the specified location.

String[] getCred1ByLocation(
    String location
    String[] class_names
)

g getScript( class_names ) throws ObjectDoesNotExist

Get the script

String[] getScript(
    String[] class_names
)

g getScriptByLocation( location, class_names ) throws ObjectDoesNotExist

Get the script. This is a location specific function, any action will operate on the specified location.

String[] getScriptByLocation(
    String location
    String[] class_names
)

g getUpdateInterval( class_names ) throws ObjectDoesNotExist

Get the interval at which cloud status is queried

Unsigned Integer[] getUpdateInterval(
    String[] class_names
)
Function Reference

getUpdateIntervalByLocation( location, class_names ) throws ObjectDoesNotExist

Get the interval at which cloud status is queried. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getUpdateIntervalByLocation(
    String location
    String[] class_names
)

renameCloudCredentials( class_names, new_names ) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, DeploymentError

Rename the named sets of cloud credentials.

void renameCloudCredentials(
    String[] class_names
    String[] new_names
)

setApiServer( class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the vcenter server hostname or IP address.

void setApiServer(
    String[] class_names
    String[] values
)

setApiServerByLocation( location, class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the vcenter server hostname or IP address. This is a location specific function, any action will operate on the specified location.

void setApiServerByLocation(
    String location
    String[] class_names
    String[] values
)

setChangeProcessTimeout( class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the amount of time change calls are allowed to take

void setChangeProcessTimeout(
    String[] class_names
    Unsigned Integer[] values
)

setChangeProcessTimeoutByLocation( location, class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the amount of time change calls are allowed to take. This is a location specific function, any action will operate on the specified location.

void setChangeProcessTimeoutByLocation(
    String location
    String[] class_names
    String[] values
)
String[] class_names
Unsigned Integer[] values
)

setCred1( class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the cloud user name
void setCred1(
    String[] class_names
    String[] values
)

setCred1ByLocation( location, class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the cloud user name This is a location specific function, any action will operate on the specified location.
void setCred1ByLocation(
    String location
    String[] class_names
    String[] values
)

setCred2( class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the cloud user password
void setCred2(
    String[] class_names
    String[] values
)

setCred2ByLocation( location, class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the cloud user password This is a location specific function, any action will operate on the specified location.
void setCred2ByLocation(
    String location
    String[] class_names
    String[] values
)

setCred3( class_names, values ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the cloud user authentication token
void setCred3(
    String[] class_names
    String[] values
)
setCred3ByLocation(location, class_names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the cloud user authentication token. This is a location specific function, any action will operate on the specified location.

```java
void setCred3ByLocation(
    String location,
    String[] class_names,
    String[] values
)
```

setScript(class_names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the script.

```java
void setScript(
    String[] class_names,
    String[] values
)
```

setScriptByLocation(location, class_names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the script. This is a location specific function, any action will operate on the specified location.

```java
void setScriptByLocation(
    String location,
    String[] class_names,
    String[] values
)
```

setUpdateInterval(class_names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the interval at which cloud status is queried.

```java
void setUpdateInterval(
    String[] class_names,
    Unsigned Integer[] values
)
```

setUpdateIntervalByLocation(location, class_names, values) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the interval at which cloud status is queried. This is a location specific function, any action will operate on the specified location.

```java
void setUpdateIntervalByLocation(
    String location,
    String[] class_names,
    Unsigned Integer[] values
)
```
**Structures**

**System.CloudCredentials.CredentialsData**

This structure contains the information needed to create CloudCredentials

```csharp
struct System.CloudCredentials.CredentialsData {
    # The user name (mandatory)
    String cred1;

    # The password (mandatory)
    String cred2;

    # The authorization token (can be empty)
    String cred3;

    # The script to use for API calls (mandatory)
    String script;

    # Time period to wait between status API calls in seconds
    Integer update_interval;
}
```

**System.Steelhead**

URI: http://soap.zeus.com/zxtm/1.0/System/Steelhead/

The System.Steelhead interface manages Riverbed Discovery Agent settings.

**Methods**

**getDiscoveryMode( traffic_managers ) throws InvalidInput, ObjectDoesNotExist**

Returns the mode used to discover Cloud Steelheads in the local data center or cloud for the supplied traffic managers.

```csharp
System.Steelhead.DiscoveryMode[] getDiscoveryMode{
    String[] traffic_managers
}
```

**getEnabled( traffic_managers ) throws InvalidInput, ObjectDoesNotExist**

Returns true if the Cloud Steelhead discovery agents on the provided traffic managers are enabled.

```csharp
Boolean[] getEnabled{
    String[] traffic_managers
}
```

**getLoadBalancingMethod( traffic_managers ) throws InvalidInput, ObjectDoesNotExist**

Returns the current load balancing method that each of the named traffic managers are using. Only required when using 'manual' mode.

```csharp
System.Steelhead.SteelheadLB[] getLoadBalancingMethod{
    String[] traffic_managers
}
getLogLevel(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Returns the level of logging used on each of the named traffic managers.

```java
System.Steelhead.LogLevel[] getLogLevel(
    String[] traffic_managers
)
```

getPortalClientID(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Returns the string used to identify the supplied traffic managers to the cloud portal. Only required in 'portal' and 'local' discovery modes.

```java
String[] getPortalClientID(
    String[] traffic_managers
)
```

getPortalClientKey(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Returns the key used to authenticate the supplied traffic managers to the cloud portal. Only required in 'portal' and 'local' discovery modes.

```java
String[] getPortalClientKey(
    String[] traffic_managers
)
```

getPortalHost(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Returns the hostname or IP address of the cloud portal to use. Only required when using the 'local' discovery mode.

```java
String[] getPortalHost(
    String[] traffic_managers
)
```

getProxyHost(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Returns the hostname or IP address of the proxy that portal communication should go through, or the empty string if no proxy should be used. Configured per traffic manager.

```java
String[] getProxyHost(
    String[] traffic_managers
)
```

getProxyPort(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Gets the current proxy server port for each of the named traffic managers. Only used if a proxy host has been specified.

```java
String[] getProxyPort(
    String[] traffic_managers
)
```

getSteelheadIPs(traffic_managers) throws InvalidInput, ObjectDoesNotExist

Gets an array of Cloud Steelhead IP addresses that each of the named traffic managers is using. Only required when using the 'manual' discovery mode.
String[][] getSteelheadIPs(
    String[] traffic_managers
)

setDiscoveryMode( traffic_managers, modes ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Sets the mode used to discover Cloud Steelheads in the local data center or cloud for the supplied traffic managers.
void setDiscoveryMode(
    String[] traffic_managers
    System.Steelhead.DiscoveryMode[] modes
)

setEnabled( traffic_managers, enabled ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Enable or disable the Cloud Steelhead discovery agent on each of the provided traffic managers.
void setEnabled(
    String[] traffic_managers
    Boolean[] enabled
)

setLoadBalancingMethod( traffic_managers, lbs ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Sets the load balancing method that each of the named traffic managers should use. Only required when using 'manual' mode.
void setLoadBalancingMethod(
    String[] traffic_managers
    System.Steelhead.SteelheadLB[] lbs
)

setLogLevel( traffic_managers, levels ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Sets the level of logging used on each of the named traffic managers.
void setLogLevel(
    String[] traffic_managers
    System.Steelhead.LogLevel[] levels
)

setPortalClientID( traffic_managers, ids ) throws InvalidInput, ObjectDoesNotExist, DeploymentError
Sets the string used to identify the supplied traffic managers to the cloud portal. Only required in 'portal' and 'local' discovery modes.
void setPortalClientID(
    String[] traffic_managers
    String[] ids
)
setPortalClientKey( traffic_managers, keys ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets the key used to authenticate the supplied traffic managers to the cloud portal. Only required in 'portal' and 'local' discovery modes.

```java
void setPortalClientKey(
    String[] traffic_managers
    String[] keys
)
```

setPortalHost( traffic_managers, hosts ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets the hostname or IP address of the cloud portal to use. Only required when using the 'local' discovery mode.

```java
void setPortalHost(
    String[] traffic_managers
    String[] hosts
)
```

setProxyHost( traffic_managers, hosts ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Set the hostname or IP address of the proxy that portal communication should go through. Set to the empty string to not use a proxy. Configured per traffic manager.

```java
void setProxyHost(
    String[] traffic_managers
    String[] hosts
)
```

setProxyPort( traffic_managers, ports ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets the current proxy server port for each of the named traffic managers. Only used if a proxy host has been specified, can be set to the empty string otherwise.

```java
void setProxyPort(
    String[] traffic_managers
    String[] ports
)
```

setSteelheadIPs( traffic_managers, ips ) throws InvalidInput, ObjectDoesNotExist, DeploymentError

Sets an array of Cloud Steelhead IP addresses that each of the named traffic managers should use. Only required when using the 'manual' discovery mode.

```java
void setSteelheadIPs(
    String[] traffic_managers
    String[][] ips
)
**System.Steelhead.DiscoveryMode**

The different modes for discovering Cloud Steelheads to forward optimized traffic to.

```csharp
enum System.Steelhead.DiscoveryMode {
    portal,
    local,
    manual
}
```

**System.Steelhead.LogLevel**

The different modes for discovering Cloud Steelheads to forward optimized traffic to.

```csharp
enum System.Steelhead.LogLevel {
    critial,
    serious,
    warning,
    notice,
    info,
    debug
}
```

**System.Steelhead.SteelheadLB**

The different modes for discovering Cloud Steelheads to forward optimized traffic to.

```csharp
enum System.Steelhead.SteelheadLB {
    round_robin,
    priority
}
```
Catalog.Aptimizer.Profile

URI: http://soap.zeus.com/zxtm/1.0/Catalog/Aptimizer/Profile/

The Catalog.Aptimizer.Profile interface allows management of Web Accelerator profiles. Using this interface, you can create, delete and rename Web Accelerator profiles, and manage their configuration.

Methods

getAptimizerProfileNames() throws LicenseError

Get the names of all the configured Web Accelerator profiles.

String[] getAptimizerProfileNames()

getBackgroundAfter( profile_names ) throws ObjectDoesNotExist, LicenseError

Get after how many milliseconds Web Accelerator should return the original server content to the client and continue optimizing data in the background.

Unsigned Integer[] getBackgroundAfter(String[] profile_names)

getBackgroundAfterByLocation( location, profile_names ) throws ObjectDoesNotExist, LicenseError

Get after how many milliseconds Web Accelerator should return the original server content to the client and continue optimizing data in the background. This is a location specific function, any action will operate on the specified location.

Unsigned Integer[] getBackgroundAfterByLocation(String location, String[] profile_names)

getBackgroundOnAdditionalResources( profile_names ) throws ObjectDoesNotExist, LicenseError

Get whether or not Web Accelerator should fetch and optimize additional resources in the background and send partially optimized content to clients until all resources are ready.

Boolean[] getBackgroundOnAdditionalResources(String[] profile_names)

getBackgroundOnAdditionalResourcesByLocation( location, profile_names ) throws ObjectDoesNotExist, LicenseError

Get whether or not Web Accelerator should fetch and optimize additional resources in the background and send partially optimized content to clients until all resources are ready. This is a location specific function, any action will operate on the specified location.

Boolean[] getBackgroundOnAdditionalResourcesByLocation(String location, String[] profile_names)
getConfig( profile_names ) throws ObjectDoesNotExist, LicenseError
Get the configuration string for the Web Accelerator profile.

```java
String[] getConfig(
    String[] profile_names
)
```

getConfigByLocation( location, profile_names ) throws ObjectDoesNotExist, LicenseError
Get the configuration string for the Web Accelerator profile. This is a location specific function, any action will operate on the specified location.

```java
String[] getConfigByLocation(
    String location,
    String[] profile_names
)
```

getMode( profile_names ) throws ObjectDoesNotExist, LicenseError
Get the mode in which Web Accelerator should run when this profile is applied

```java
Catalog.Aptimizer.Profile.Mode[] getMode(
    String[] profile_names
)
```

getModeByLocation( location, profile_names ) throws ObjectDoesNotExist, LicenseError
Get the mode in which Web Accelerator should run when this profile is applied. This is a location specific function, any action will operate on the specified location.

```java
Catalog.Aptimizer.Profile.Mode[] getModeByLocation(
    String location,
    String[] profile_names
)
```

getShowInfoBar( profile_names ) throws ObjectDoesNotExist, LicenseError
Get whether or not the Web Accelerator information bar should be displayed on accelerated web pages

```java
Boolean[] getShowInfoBar(
    String[] profile_names
)
```

getShowInfoBarByLocation( location, profile_names ) throws ObjectDoesNotExist, LicenseError
Get whether or not the Web Accelerator information bar should be displayed on accelerated web pages. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getShowInfoBarByLocation(
    String location,
    String[] profile_names
)
```
setBackgroundAfter( profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set after how many milliseconds Web Accelerator should return the original server content to the client and continue optimizing data in the background.

```java
void setBackgroundAfter(
    String[] profile_names
    Unsigned Integer[] values
)
```

setBackgroundAfterByLocation( location, profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set after how many milliseconds Web Accelerator should return the original server content to the client and continue optimizing data in the background. This is a location specific function, any action will operate on the specified location.

```java
void setBackgroundAfterByLocation(
    String location
    String[] profile_names
    Unsigned Integer[] values
)
```

setBackgroundOnAdditionalResources( profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set whether or not Web Accelerator should fetch and optimize additional resources in the background and send partially optimized content to clients until all resources are ready.

```java
void setBackgroundOnAdditionalResources(
    String[] profile_names
    Boolean[] values
)
```

setBackgroundOnAdditionalResourcesByLocation( location, profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set whether or not Web Accelerator should fetch and optimize additional resources in the background and send partially optimized content to clients until all resources are ready. This is a location specific function, any action will operate on the specified location.

```java
void setBackgroundOnAdditionalResourcesByLocation(
    String location
    String[] profile_names
    Boolean[] values
)
```

setConfig( profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set the configuration string for the Web Accelerator profile.

```java
void setConfig(
    String[] profile_names
    String[] values
)
setConfigByLocation( location, profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set the configuration string for the Web Accelerator profile. This is a location specific function, any action will operate on the specified location.

```java
void setConfigByLocation(
    String location,
    String[] profile_names,
    String[] values
)
```

setMode( profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set the mode in which Web Accelerator should run when this profile is applied.

```java
void setMode(
    String[] profile_names,
    Catalog.Aptimizer.Profile.Mode[] values
)
```

setModeByLocation( location, profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set the mode in which Web Accelerator should run when this profile is applied. This is a location specific function, any action will operate on the specified location.

```java
void setModeByLocation(
    String location,
    String[] profile_names,
    Catalog.Aptimizer.Profile.Mode[] values
)
```

setShowInfoBar( profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set whether or not the Web Accelerator information bar should be displayed on accelerated web pages.

```java
void setShowInfoBar(
    String[] profile_names,
    Boolean[] values
)
```

setShowInfoBarByLocation( location, profile_names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidOperation, DeploymentError, LicenseError

Set whether or not the Web Accelerator information bar should be displayed on accelerated web pages. This is a location specific function, any action will operate on the specified location.

```java
void setShowInfoBarByLocation(
    String location,
    String[] profile_names,
    Boolean[] values
)
```
Enumerations

Catalog.Aptimizer.Profile.Mode

```java
catalog.Aptimizer.Profile.Mode {
    # Off - Acceleration is disabled, but requests for Web Accelerator resources
    # are served
    idle,

    # Stealth - Acceleration is controlled by a cookie
    stealth,

    # On - Web Accelerator acceleration is enabled
    active
}
```

Catalog.Kerberos.Principals

URI: http://soap.zeus.com/zxtm/1.0/Catalog/Kerberos/Principals/

The Catalog.Kerberos.Principal interface allows management of Kerberos Principals. Using this interface, you can create, delete and rename Kerberos principals, and manage their configuration.

Methods

addKDCs( principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add a host/port pair to the lists of KDCs, for each of the named principals

```java
void addKDCs{
    String[] principal_names
    String[][][] values
}
```

addKDCsByLocation( location, principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Add a host/port pair to the lists of KDCs, for each of the named principals This is a location specific function, any action will operate on the specified location.

```java
void addKDCsByLocation{
    String location
    String[] principal_names
    String[][][] values
}
```

addPrincipal( principal_names, principal_parameters ) throws InvalidObjectName, InvalidInput, ObjectAlreadyExists, DeploymentError, LicenseError

Add new Kerberos principals.

```java
void addPrincipal{
    String[] principal_names
    Catalog.Kerberos.Principals.PrincipalParameter[] principal_parameters
```
copyPrincipal(principal_names, new_names) throws ObjectAlreadyExists, InvalidObjectName, ObjectDoesNotExist, DeploymentError, LicenseError

Copy the named Kerberos principals.

```java
void copyPrincipal(
    String[] principal_names
    String[] new_names
)
```

deletePrincipal(principal_names) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, LicenseError

Delete the named Kerberos principals.

```java
void deletePrincipal(
    String[] principal_names
)
```

getKDCs(principal_names) throws ObjectDoesNotExist, LicenseError

Get the explicit list of Kerberos KDCs, for each of the named principals

```java
String[][] getKDCs(
    String[] principal_names
)
```

getKDCsByLocation(location, principal_names) throws ObjectDoesNotExist, LicenseError

Get the explicit list of Kerberos KDCs, for each of the named principals. This is a location specific function, any action will operate on the specified location.

```java
String[][] getKDCsByLocation(
    String location
    String[] principal_names
)
```

getKRB5Conf(principal_names) throws ObjectDoesNotExist, LicenseError

Get the name of the optional Kerberos configuration file.

```java
String[] getKRB5Conf(
    String[] principal_names
)
```

getKRB5ConfByLocation(location, principal_names) throws ObjectDoesNotExist, LicenseError

Get the name of the optional Kerberos configuration file. This is a location specific function, any action will operate on the specified location.

```java
String[] getKRB5ConfByLocation(
    String location
    String[] principal_names
)
```
getKeytab( principal_names ) throws ObjectDoesNotExist, LicenseError
Get the name of the associated Kerberos keytab file

```java
String[] getKeytab(
    String[] principal_names
)
```

getKeytabByLocation( location, principal_names ) throws ObjectDoesNotExist, LicenseError
Get the name of the associated Kerberos keytab file This is a location specific function, any action will operate on the specified location.

```java
String[] getKeytabByLocation(
    String location
    String[] principal_names
)
```

getPrincipalNames()
Get the names of all the configured Kerberos principals.

```java
String[] getPrincipalNames()
```

getRealm( principal_names ) throws ObjectDoesNotExist, LicenseError
Get the Kerberos realm

```java
String[] getRealm(
    String[] principal_names
)
```

getRealmByLocation( location, principal_names ) throws ObjectDoesNotExist, LicenseError
Get the Kerberos realm This is a location specific function, any action will operate on the specified location.

```java
String[] getRealmByLocation(
    String location
    String[] principal_names
)
```

getService( principal_names ) throws ObjectDoesNotExist, LicenseError
Get the service name aspect of the Kerberos principal name the traffic manager should use to authenticate itself

```java
String[] getService(
    String[] principal_names
)
```

getServiceByLocation( location, principal_names ) throws ObjectDoesNotExist, LicenseError
Get the service name aspect of the Kerberos principal name the traffic manager should use to authenticate itself This is a location specific function, any action will operate on the specified location.

```java
String[] getServiceByLocation(
    String location
)
```
String[] principal_names
}

removeKDCs(principal_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Remove a host/port pair from the lists of KDCs, for each of the named principals
void removeKDCs(
    String[] principal_names
    String[][] values
)

removeKDCsByLocation(location, principal_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Remove a host/port pair from the lists of KDCs, for each of the named principals This is a location specific function, any action will operate on the specified location.
void removeKDCsByLocation(
    String location
    String[] principal_names
    String[][] values
)

renamePrincipal(principal_names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, InvalidOperation, DeploymentError, LicenseError

Rename the named Kerberos principals.
void renamePrincipal(
    String[] principal_names
    String[] new_names
)

setKDCs(principal_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the explicit list of Kerberos KDCs, for each of the named principals
void setKDCs(
    String[] principal_names
    String[][] values
)

setKDCsByLocation(location, principal_names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the explicit list of Kerberos KDCs, for each of the named principals This is a location specific function, any action will operate on the specified location.
void setKDCsByLocation(
    String location
    String[] principal_names
    String[][] values
)
setKRB5Conf( principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the name of the optional Kerberos configuration file

```java
void setKRB5Conf(
    String[] principal_names
    String[] values
)
```

setKRB5ConfByLocation( location, principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the name of the optional Kerberos configuration file This is a location specific function, any action will operate on the specified location.

```java
void setKRB5ConfByLocation(
    String location
    String[] principal_names
    String[] values
)
```

setKeytab( principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the name of the associated Kerberos keytab file

```java
void setKeytab(
    String[] principal_names
    String[] values
)
```

setKeytabByLocation( location, principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the name of the associated Kerberos keytab file This is a location specific function, any action will operate on the specified location.

```java
void setKeytabByLocation(
    String location
    String[] principal_names
    String[] values
)
```

setRealm( principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the Kerberos realm

```java
void setRealm(
    String[] principal_names
    String[] values
)
```

setRealmByLocation( location, principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the Kerberos realm This is a location specific function, any action will operate on the specified location.

```java
void setRealmByLocation(
```
String location
String[] principal_names
String[] values
}

setService( principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the service name aspect of the Kerberos principal name the traffic manager should use to authenticate itself.

void setService(
    String[] principal_names
    String[] values
)

setServiceByLocation( location, principal_names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError, LicenseError

Set the service name aspect of the Kerberos principal name the traffic manager should use to authenticate itself. This is a location specific function, any action will operate on the specified location.

void setServiceByLocation(
    String location
    String[] principal_names
    String[] values
)

Structures

Catalog.Kerberos.Principals.PrincipalParameter

This structure contains the required configuration values for a Kerberos principal.

struct Catalog.Kerberos.Principals.PrincipalParameter  {
    # The service name part of the Kerberos principal name the traffic manager
    # should use to authenticate itself.
    String service;

    # The Kerberos realm where the principal belongs.
    String realm;

    # The name of the Kerberos keytab file containing suitable credentials to
    # authenticate as the specified Kerberos principal.
    String keytab;
}

Catalog.Kerberos.KeyTabs

URI: http://soap.zeus.com/zxtm/1.0/Catalog/Kerberos/KeyTabs/

The Catalog.Kerberos.KeyTabs interface allows management of the Kerberos keytabs stored in the conf/kerberos/keytabs directory. Kerberos keytabs contain credentials for any number of Kerberos principals.
**Methods**

**deleteFile( names ) throws ObjectDoesNotExist, LicenseError**

Delete the named Kerberos keytabs from the conf/kerberos/keytabs directory.

```java
void deleteFile(
    String[] names
)
```

**getFileName()**

Get the names of all the Kerberos keytabs stored in the conf/kerberos/keytabs directory.

```java
String[] getFileNames()
```

**uploadFile( name, content ) throws InvalidObjectName, LicenseError**

Uploads a new Kerberos keytab into the conf/kerberos/keytabs, overwriting the keytab if it already exists.

```java
void uploadFile(
    String name,
    Binary Data content
)
```

---

**Catalog.Kerberos.Krb5Confs**

URI: http://soap.zeus.com/zxtm/1.0/Catalog/Kerberos/Krb5Confs/

The Catalog.Kerberos.Krb5Confs interface allows management of the Kerberos configuration files stored in the conf/kerberos/krb5confs directory. These configuration can, optionally, be used for the configuration of a Kerberos principal.

**Methods**

**deleteFile( names ) throws ObjectDoesNotExist, LicenseError**

Delete the named configuration files from the conf/kerberos/krb5confs directory.

```java
void deleteFile(
    String[] names
)
```

**downloadFile( name ) throws ObjectDoesNotExist, LicenseError**

Download the named configuration file from the conf/kerberos/krb5confs directory

```java
Binary Data downloadFile(
    String name
)
```

**getFileName()**

Get the names of all the Kerberos configuration files stored in the conf/kerberos/krb5confs directory.
uploadFile( name, content ) throws InvalidObjectName, LicenseError

Uploads a new Kerberos configuration file into the conf/kerberos/krb5conf, overwriting the configuration file if it already exists.

```java
void uploadFile(
    String name
    Binary Data content
)
```

---

**BGPNeighbors**

URI: http://soap.zeus.com/zxtm/1.0/BGPNeighbors/

The BGPNeighbor interface allows management of BGP neighbor objects. Using this interface, you can create, delete and rename BGP neighbor objects, and manage their configuration.

**Methods**

**addBGPNeighbor( names, details ) throws ObjectAlreadyExists, InvalidObjectName, DeploymentError, InvalidInput, InvalidOperation**

Add the new named BGP neighbors, using the provided details.

```java
void addBGPNeighbor(
    String[] names
    BGPNeighbors.Details[] details
)
```

**addTrafficManager( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidInput**

Add new traffic managers to each of the named neighbors.

```java
void addTrafficManager(
    String[] names
    String[][] values
)
```

**deleteBGPNeighbor( names ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError, InvalidOperation**

Delete the named BGP neighbors.

```java
void deleteBGPNeighbor(
    String[] names
)
```

**getAddress( names )**

Get the address of the BGP neighbor.

```java
String[] getAddress()
```
getAdvertisementInterval( names )
Get the minimum interval between the sending of BGP routing updates.

Unsigned Integer[] getAdvertisementInterval(
    String[] names
)

getAsNumber( names )
Get the AS number for the AS which the neighbor is a member of

Unsigned Integer[] getAsNumber(
    String[] names
)

getAuthenticationPassword( names )
Get the password to be used for BGP authentication.

String[] getAuthenticationPassword(
    String[] names
)

getBGPNeighborNames()
Get the names of all of the configured BGP neighbors.

String[] getBGPNeighborNames()

getHoldtime( names )
Get the period after which the BGP session with the neighbour is deemed to have become idle - and requires re-establishment - if the neighbour falls silent.

Unsigned Integer[] getHoldtime(
    String[] names
)

getKeepalive( names )
Get the interval at which BGP keepalive messages are sent to the BGP neighbors, to keep the mutual BGP session established.

Unsigned Integer[] getKeepalive(
    String[] names
)

getTrafficManager( names ) throws ObjectDoesNotExist
Get the traffic managers that will establish a BGP session with this neighbor.

String[][] getTrafficManager(
    String[] names
)
removeTrafficManager( names, values ) throws ObjectDoesNotExist, DeploymentError, InvalidOperation

Remove the named traffic managers from each named neighbor.

```java
void removeTrafficManager(
    String[] names
    String[][] values
)
```

renameBGPNeighbor( names, new_names ) throws ObjectDoesNotExist, DeploymentError, InvalidInput, InvalidOperation

Rename each of the named BGP neighbors.

```java
void renameBGPNeighbor(
    String[] names
    String[] new_names
)
```

setAddress( names, values ) throws InvalidInput, DeploymentError

Set the address of the BGP neighbor.

```java
void setAddress(
    String[] names
    String[] values
)
```

setAdvertisementInterval( names, values ) throws InvalidInput, DeploymentError

Set the minimum interval between the sending of BGP routing updates.

```java
void setAdvertisementInterval(
    String[] names
    Unsigned Integer[] values
)
```

setAsNumber( names, values ) throws InvalidInput, DeploymentError

Set the AS number for the AS which the neighbor is a member of

```java
void setAsNumber(
    String[] names
    Unsigned Integer[] values
)
```

setAuthenticationPassword( names, values ) throws InvalidInput, DeploymentError

Set the password to be used for BGP authentication.

```java
void setAuthenticationPassword(
    String[] names
    String[] values
)
```

setHoldtime( names, values ) throws InvalidInput, DeploymentError

Set the period after which the BGP session with the neighbour is deemed to have become idle - and requires re-establishment - if the neighbour falls silent.
Function Reference

BGPNeighbors

void setHoldtime(
    String[] names
    Unsigned Integer[] values
)

setKeepalive( names, values ) throws InvalidInput, DeploymentError

Set the interval at which BGP keepalive messages are sent to the BGP neighbors, to keep the mutual BGP session established.

void setKeepalive(
    String[] names
    Unsigned Integer[] values
)

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

Set the traffic managers that will establish a BGP session with this neighbor.

void setTrafficManager(
    String[] names
    String[][] values
)

Structures

BGPNeighbors.Details

This structure contains the basic details of a BGP Neighbor: the configuration needed to establish a BGP session with the neighbor, and the traffic managers which will use the neighbor. It is used when creating a new BGP Neighbor configuration object.

struct BGPNeighbors.Details {
    # The IPv4 address of the BGP neighbor.
    String address;

    # The number of the AS in which the BGP neighbor is operating.
    Integer as_number;

    # The required interval between keepalive messages for BGP sessions with the neighbor.
    Integer keepalive;

    # The maximum interval between keepalive messages for BGP sessions with the neighbor to remain established.
    Integer holdtime;

    # The minimum interval between successive advertisements being sent during a BGP session with the neighbor.
    Integer advertisement_interval;

    # The password shared with the BGP neighbor to authenticate BGP sessions.
    String authentication_password;

    # The names of the traffic managers that will establish BGP sessions with the neighbor.
    String[] machines;
}
Analytics.LogExport

URI: http://soap.zeus.com/zxtm/1.0/Analytics/LogExport/

The Analytics.LogExport interface allows management of log files which should be exported to an analytics engine.

**Methods**

`addFiles( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError`

Add the set of files to export as part of this category, specified as a list of glob patterns.

```java
void addFiles(
    String[] names
    String[][] values
)
```

`addFilesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError`

Add the set of files to export as part of this category, specified as a list of glob patterns. This is a location specific function, any action will operate on the specified location.

```java
void addFilesByLocation(
    String location
    String[] names
    String[][] values
)
```

`addLogExport( names ) throws InvalidObjectName, InvalidInput, ObjectAlreadyExists, DeploymentError`

Add new log export categories.

```java
void addLogExport(
    String[] names
)
```

`copyLogExport( old_names, new_names ) throws ObjectAlreadyExists, InvalidObjectName, ObjectDoesNotExist, DeploymentError`

Copy the named log export categories.

```java
void copyLogExport(
    String[] old_names
    String[] new_names
)
```

`deleteLogExport( names ) throws ObjectDoesNotExist, ObjectInUse, DeploymentError`

Delete the named log export categories.

```java
void deleteLogExport(
```
deleteMetadata( names, metadata_names ) throws ObjectDoesNotExist

Deletes the named metadata values.

```java
void deleteMetadata{
    String[] names
    String[] metadata_names
}
```

getApplianceOnly( names ) throws ObjectDoesNotExist

Get whether entries from the specified log files should be exported only from appliances.

```java
Boolean[] getApplianceOnly{
    String[] names
}
```

getApplianceOnlyByLocation( location, names ) throws ObjectDoesNotExist

Get whether entries from the specified log files should be exported only from appliances. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getApplianceOnlyByLocation{
    String location
    String[] names
}
```

getBuiltIn( names ) throws ObjectDoesNotExist

Get whether this configuration is built-in. Editing and deletion of built in configurations is restricted.

```java
Boolean[] getBuiltIn{
    String[] names
}
```

getEnabled( names ) throws ObjectDoesNotExist

Get whether to export entries from the log files included in this category.

```java
Boolean[] getEnabled{
    String[] names
}
```

getEnabledByLocation( location, names ) throws ObjectDoesNotExist

Get whether to export entries from the log files included in this category. This is a location specific function, any action will operate on the specified location.

```java
Boolean[] getEnabledByLocation{
    String location
    String[] names
}
```

getFiles( names ) throws ObjectDoesNotExist

Get the set of files to export as part of this category, specified as a list of glob patterns.

```java
String[][] getFiles{
```
String[] names
)

getFilesByLocation( location, names ) throws ObjectDoesNotExist
Get the set of files to export as part of this category, specified as a list of glob patterns. This is a location specific function, any action will operate on the specified location.
String[][] getFilesByLocation(?
    String location
    String[] names
)

getHistory( names ) throws ObjectDoesNotExist
Get how much historic log activity should be exported.
Analytics.LogExport.History[] getHistory(?
    String[] names
)

getHistoryByLocation( location, names ) throws ObjectDoesNotExist
Get how much historic log activity should be exported. This is a location specific function, any action will operate on the specified location.
Analytics.LogExport.History[] getHistoryByLocation(?
    String location
    String[] names
)

getHistoryPeriod( names ) throws ObjectDoesNotExist
Get the number of days of historic log entries that should be exported.
Unsigned Integer[] getHistoryPeriod(?
    String[] names
)

getHistoryPeriodByLocation( location, names ) throws ObjectDoesNotExist
Get the number of days of historic log entries that should be exported. This is a location specific function, any action will operate on the specified location.
Unsigned Integer[] getHistoryPeriodByLocation(?
    String location
    String[] names
)

getLogExportNames()
Get the names of all the configured log export categories.
String[] getLogExportNames()

getMetadata( names ) throws ObjectDoesNotExist
Get the metadata.
Analytics.LogExport.Metadata[][] getMetadata(?
    String[] names
)
getNote(names) throws ObjectDoesNotExist

Get a description of this category of log files.

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

generateByLocation(location, names) throws ObjectDoesNotExist

Get a description of this category of log files. This is a location specific function, any action will operate on the specified location.

```java
String[] getNoteByLocation(
    String location
    String[] names
)
```

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

Remove the set of files to export as part of this category, specified as a list of glob patterns.

```java
void removeFiles(
    String[] names
    String[][] values
)
```

removeFilesByLocation(location, names, values) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Remove the set of files to export as part of this category, specified as a list of glob patterns. This is a location specific function, any action will operate on the specified location.

```java
void removeFilesByLocation(
    String location
    String[] names
    String[][] values
)
```

renameLogExport(old_names, new_names) throws ObjectAlreadyExists, ObjectDoesNotExist, InvalidObjectName, InvalidOperation, DeploymentError

Rename the named log export categories.

```java
void renameLogExport(
    String[] old_names
    String[] new_names
)
```

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

Set whether entries from the specified log files should be exported only from appliances.

```java
void setApplianceOnly(
    String[] names
    Boolean[] values
)
setApplianceOnlyByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether entries from the specified log files should be exported only from appliances. This is a location specific function, any action will operate on the specified location.

```java
void setApplianceOnlyByLocation(
    String location
    String[] names
    Boolean[] values
)
```

setBuiltIn( names, values ) throws InvalidOperation

Set whether this configuration is built-in. Editing and deletion of built in configurations is restricted.

```java
void setBuiltIn(
    String[] names
    Boolean[] values
)
```

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

Set whether to export entries from the log files included in this category.

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

setEnabledByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set whether to export entries from the log files included in this category. This is a location specific function, any action will operate on the specified location.

```java
void setEnabledByLocation(
    String location
    String[] names
    Boolean[] values
)
```

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

Set the set of files to export as part of this category, specified as a list of glob patterns.

```java
void setFiles(
    String[] names
    String[][] values
)
Function Reference

setFilesByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the set of files to export as part of this category, specified as a list of glob patterns. This is a location specific function, any action will operate on the specified location.

```java
void setFilesByLocation(
    String location,
    String[] names,
    String[][] values
)
```

setHistory( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set how much historic log activity should be exported.

```java
void setHistory(
    String[] names,
    Analytics.LogExport.History[] values
)
```

setHistoryByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set how much historic log activity should be exported. This is a location specific function, any action will operate on the specified location.

```java
void setHistoryByLocation(
    String location,
    String[] names,
    Analytics.LogExport.History[] values
)
```

setHistoryPeriod( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the number of days of historic log entries that should be exported.

```java
void setHistoryPeriod(
    String[] names,
    Unsigned Integer[] values
)
```

setHistoryPeriodByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError
Set the number of days of historic log entries that should be exported. This is a location specific function, any action will operate on the specified location.

```java
void setHistoryPeriodByLocation(
    String location,
    String[] names,
    Unsigned Integer[] values
)
```
setMetadata( names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set the metadata.

```java
void setMetadata(
    String[] names
    Analytics.LogExport.Metadata[][] values
)
```

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

Set a description of this category of log files.

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

setNoteByLocation( location, names, values ) throws ObjectDoesNotExist, InvalidInput, DeploymentError

Set a description of this category of log files. This is a location specific function, any action will operate on the specified location.

```java
void setNoteByLocation(
    String location
    String[] names
    String[] values
)
```

Structures

Analytics.LogExport.Metadata

A named metadata item for a log export category

```java
struct Analytics.LogExport.Metadata  {
    # The name of the metadata item.
    String name;

    # The value of the metadata item.
    String value;
}
```

Enumerations

Analytics.LogExport.History

```java
enum Analytics.LogExport.History {
    # Do not export any historic entries
    none,

    # Export all historic entries
    all,

    # Export recent historic entries, according to the 'history_period' setting
```
Custom

URI: http://soap.zeus.com/zxtm/1.0/Custom/

The Custom interface allows management of custom configuration sets. Using this interface, you can create, delete and rename custom configuration sets and the values they contain.

Methods

addSets( names ) throws ObjectAlreadyExists, InvalidObjectName

Add new custom configuration sets

void addSets(
    String[] names
)

addStringLists( names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidObjectName

Appends the specified lists of strings to the current lists in the named custom configuration sets.

void addStringLists(
    String[] names
    Custom.StringList[][] values
)

copySets( names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName

Copy custom configuration sets.

void copySets(
    String[] names
    String[] new_names
)

deleteSets( names ) throws ObjectDoesNotExist, InvalidObjectName

Delete custom configuration sets.

void deleteSets(
    String[] names
)

getSetNames()

Get the names of all custom configuration sets.

String[] getSetNames()
getStringLists( names ) throws ObjectDoesNotExist, InvalidObjectName

 Gets all the lists of strings in the named custom configuration sets.

 Custom.StringList[][] getStringLists{
     String[] names
 }

removeStringListItems( names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidObjectName

 Removes all occurrences of the specified strings from the lists in the named custom configuration sets.

 void removeStringListItems{
     String[] names
     Custom.StringList[][] values
 }

removeStringLists( names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidObjectName

 Removes the specified lists of strings from the named custom configuration sets.

 void removeStringLists{
     String[] names
     String[][] values
 }

renameSets( names, new_names ) throws ObjectDoesNotExist, ObjectAlreadyExists, InvalidObjectName

 Rename custom configuration sets.

 void renameSets{
     String[] names
     String[] new_names
 }

setStringLists( names, values ) throws ObjectDoesNotExist, InvalidInput, InvalidObjectName

 Sets the specified lists of strings in the named custom configuration sets.

 void setStringLists{
     String[] names
     Custom.StringList[][] values
 }

Structures

Custom.StringList

 A named list of strings in a custom configuration set

 struct Custom.StringList {
     # The name of the list of strings.
     String name;

     # The list of strings.
SOAPS 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.

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.

```java
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.

```java
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 '/').

```c
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.

```c
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.

```c
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.

```c
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.

```java
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.

```java
struct ObjectInUse {
    # A human readable string describing the error
    String errmsg;

    # The name of the object that caused the fault
    String object;
}
```