

Traverse™

5.0



© 2009 Zyrion, Inc. All rights reserved. Zyrion, the Zyrion logo, Traverse are registered trademarks of Zyrion, Inc. and/or its affiliates in the United States and/or other countries. All other registered and unregistered trademarks herein are the sole property of their respective owners. Zyrion, Inc. reserves the right, at its sole discretion, to make changes at any time in its technical information and specifications, and service and support programs.

Zyrion, Inc. End User License Agreement

ZYRION, INC., ON BEHALF OF ITSELF AND ITS SUBSIDIARIES AND AFFILIATES, ("ZYRION") WILL LICENSE PRODUCTS TO YOU ONLY IF YOU ACCEPT THIS END USER LICENSE AGREEMENT AND APPLICABLE SUPPLEMENTAL TERMS (COLLECTIVELY "AGREEMENT"). CAREFULLY READ THESE TERMS BEFORE USING THE PRODUCTS. By clicking the "I accept" button below or by installing or using the Software, you have indicated that you understand this Agreement and accept all of its terms. If you do not accept all the terms of this Agreement, click on the button that indicates that you do not accept the terms of this Agreement and do not install the Software. Some third party materials may be included in the Products and subject to other terms found in the "Read Me" or "About" file. By using the Software, you agree to comply with such terms.

Definitions

"APIs" mean the software application interfaces and workflow methods made generally available by Zyrion in certain Products to enable integration, implementation, and interoperability with third party hardware and software.

"Documentation" means installation guides and operation manuals provided with the Product on its original date of shipment in printed, electronic, or online form.

"Hardware" means hardware products generally available on Zyrion's price list.

"Zyrion Quotation" means the document under which Zyrion offers for sale and license its Products and associated services.

"Product" means Software and Hardware provided by Zyrion or Zyrion's authorized reseller or distributor.

"Software" means Zyrion proprietary programs in object code and the firmware contained on the Hardware. The term Software does not include APIs.

"Software Development Kit" or "SDK" means the Zyrion API, together with the Documentation, any sample code, and any sample applications provided with the API.

1. License. Zyrion grants you a limited, non-exclusive, non-transferable license to use the Software for which you paid the applicable license fee and the Documentation, subject to the terms set forth in this Agreement and the technical requirements set forth in the Documentation.

2. License Restrictions.

(a) You may:

- i. use the Software on a single computer;
- ii. make a copy of the Software for archival or backup purposes only ("Copy"). The Copy may not be used to implement a fault tolerant environment, redundant environment, or contingency environment; and
- iii. use the Software and Documentation for your own internal business purposes.

(b) Zyrion retains all right, title, and interest in and to the Software (which includes all corrections, modifications, enhancements, updates, and upgrades), Documentation, and Copies, and all patents, copyrights, trade secrets, trademarks, and other intellectual property rights therein. Zyrion retains all rights to the intellectual property associated with the Hardware except as expressly granted in this Agreement, and the above Software restrictions will apply to Hardware to the extent applicable. The Software, Documentation, and Copies are protected under copyright laws, and any permitted Copies must include all copyright, government-restricted rights, and other proprietary notices or legends included on the Software when it was shipped to you. Without limiting the generality of the foregoing, you, your employees, and your consultants will not and will not authorize or permit any third party to:

- i. copy or reproduce any part of the Software or Documentation except in loading the Software into, or making a Copy for, the single computer as permitted above;
- ii. transfer the Software from one machine to another or share the Software between or among multiple machines through networking or other communication packages;
- iii. sell, market, distribute, sublicense, lease, provide timeshares, rent, or grant other rights in the Software to others or permit third parties to access the Software, without the written consent of Zyrion;
- iv. use the Software or any portions thereof for which you have not paid the applicable fee;
- v. modify, develop, port, translate, localize, reverse engineer, de-compile, disassemble, or create derivative works based on the Software, except to the extent expressly permitted by applicable law and solely to extent the parties shall not be permitted by that applicable law to exclude or limit such rights.

Additional restrictions are as set forth in the applicable Supplemental Terms and terms governing open source software that are located in the Documentation.

3. Evaluation Copies

Zyriion may distribute Product for evaluation ("Evaluation Product"). While the Evaluation Product is identical to the commercial Product, the Evaluation Product may not be used for production purposes, and, for Software, contains a license key that disables the Software after 30 days and which will render the Evaluation Product unusable. If, after using the Evaluation Product, you wish to continue such use, you must purchase the Product. Upon payment of the applicable fee, Product that can be used for production purposes will be provided to you, along with the Documentation. Zyriion does not offer Maintenance on Evaluation Products.

4. License Term. The license is effective until terminated. You may terminate the license at any time by destroying the Software, Documentation, and Copies, and providing written certification to Zyriion that all of the foregoing have been destroyed. This license will also terminate if you or your employees or consultants fail to comply with any terms of this Agreement. You agree that upon such termination you will either return the Software, Documentation, and Copies or, with Zyriion's prior consent, destroy the Software, Documentation, and Copies.

5. Confidentiality. The Product contains valuable trade secrets of Zyriion and constitutes confidential information of Zyriion and its licensors. You agree to protect the confidentiality of the Product with the same degree of care by which you protect your own such confidential information, but no less than reasonable care. Accordingly, you will not provide access to or disclose the Product or Copy to any third party without the prior written consent of a duly authorized U.S. Zyriion corporate officer.

6. Limited Warranty. Zyriion warrants that the media on which the Software is recorded will be free from defects in materials and workmanship under normal use and service for a period of 90 days from the original date of shipment of the Software ("Media Warranty Period"). Zyriion warrants that the Software for a period of 90 days ("Software Warranty Period") and the Hardware for a period of 12 months ("Hardware Warranty Period"), in either case from its original date of shipment, will substantially conform to the Documentation.

If, during (a) the Media Warranty Period, a defect in the media occurs and is reported to Zyriion, the media may be returned to Zyriion, and Zyriion will replace the media without charge to you, or (b) the Software Warranty Period or Hardware Warranty Period, a failure of the Software or Hardware to conform as warranted occurs and is reported to Zyriion, Zyriion, at its option, will use commercially reasonable efforts to repair or replace the non-conforming Software or Hardware.

The foregoing warranties will apply provided you give Zyriion prompt written notice of the material defect or nonconformity within the warranty period specified above and return the defective media or non-conforming Software or Hardware to Zyriion.

7. Warranty Limit. The warranty set forth in Section 6 does not apply to any failure of the Software or Hardware caused by (a) your failure to follow Zyriion's installation, operation, or maintenance instructions or procedures; (b) your mishandling, misuse, negligence, or improper installation, de-installation, storage, servicing, or operation of the Product; (c) unauthorized modifications or repairs; and (d) power failures or surges, fire, flood, accident, actions of third parties, or other events outside Zyriion's reasonable control.

Zyriion cannot and does not warrant the performance or results that may be obtained by using the Products, nor does Zyriion warrant that Products are appropriate for your purposes or error-free.

EXCEPT AS OTHERWISE PROVIDED IN SECTION 6, THE WARRANTY SET FORTH IN SECTION 6 IS YOUR SOLE AND EXCLUSIVE REMEDY AND ZYRIION'S ENTIRE LIABILITY FOR DEFECTIVE MEDIA OR NON-CONFORMING PRODUCTS AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT.

8. Liability Limit. TO THE MAXIMUM EXTENT PERMITTED BY LAW, EXCEPT IN THE EVENT OF YOUR (a) BREACH OF THE LICENSE GRANT, (b) VIOLATION OF THE RESTRICTIONS SET FORTH IN SECTION 2 OF THIS AGREEMENT, OR (c) BREACH OF CONFIDENTIALITY, IN NO EVENT WILL EITHER PARTY OR ITS LICENSORS OR SUPPLIERS BE LIABLE FOR SPECIAL, INDIRECT, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS, LOSS OF REVENUE, LOSS OF USE, LOSS OF DATA, BUSINESS INTERRUPTION, OR THE COST OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. IN NO EVENT WILL THE CUMULATIVE LIABILITY OF ZYRIION EXCEED THE AMOUNTS PAID TO ZYRIION FOR THE APPLICABLE PRODUCT OR SERVICE THAT GAVE RISE TO SUCH CLAIM.

NOTWITHSTANDING ANYTHING TO THE CONTRARY IN THE FOREGOING, IN THE CASE OF SDKs AND EVALUATION PRODUCTS, ZYRIION'S CUMULATIVE LIABILITY WILL NOT EXCEED ONE HUNDRED U.S. DOLLARS.

THE FOREGOING LIMITS WILL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN. THE LIMITATIONS OF LIABILITY SET FORTH IN THIS AGREEMENT ARE CUMULATIVE AND ARE INTENDED AND ACKNOWLEDGED BY END USER TO BENEFIT ZYRIION AND ITS THIRD PARTY SUPPLIERS.

ZYRIION'S LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM THE NEGLIGENCE OF ZYRIION OR THAT OF ITS EMPLOYEES WHICH MAY NOT BY APPLICABLE LAW BE EXCLUDED IS NOT EXCLUDED BY THIS AGREEMENT BUT IS RATHER LIMITED TO THE MAXIMUM EXTENT PERMISSIBLE BY APPLICABLE LAW.

Some jurisdictions do not allow the exclusion or limitation of direct, incidental, or consequential damages, so the above limitations may not apply to you.

- 9. Audit Rights.** Zyriion may conduct, during normal business hours, an audit of your applicable records to verify the number of copies of the Software in use and the host processors on which such copies are installed.
- 10. Compliance with Laws.** You agree to comply, at your own expense, with all laws, regulations, rules, and ordinances of any governmental body, department, or agency that apply to or result from your obligations under this Agreement. To the extent that the Software is subject to U.S. export controls, you agree to comply fully, at your own expense, with all applicable U.S. laws and regulations governing the export, destination, ultimate end user, and other restrictions relating to the Software.
- 11. Survival.** Sections 2(b) and 5-17 will survive termination of this Agreement.
- 12. Assignment.** You will not directly or indirectly sell, transfer, assign, or delegate in whole or in part this Agreement, or any rights, duties, obligations, or liabilities under this Agreement, to any third party, including to any affiliated entity, without the prior written consent of Zyriion.
- 13. U. S. Government Restricted Rights.** All Zyriion Software, including the Documentation and technical data, sold or delivered pursuant to this Agreement for Government use are commercial as defined in Federal Acquisition Regulation ("FAR") 2.101 and any supplement and further is provided with RESTRICTED RIGHTS. All Software was fully developed at private expense. Use, duplication, release, modification, transfer, or disclosure (for purposes of this section, "Use") of the Software is restricted by the terms of this Agreement and further restricted in accordance with FAR 52.227-14 for civilian Government agency purposes and 252.227-7015 of the Defense Federal Acquisition Regulations Supplement ("DFARS") for military Government agency purposes, or the similar acquisition regulations of other applicable Government organizations, as applicable and amended. The Use of Software and the Product is restricted by the terms of this Agreement, in accordance with DFARS Section 227.7202 and FAR Section 12.212. All other Use is prohibited except as described herein.
- 14. General.** This Agreement, any attachments, and Zyriion's Quotation constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior agreements, arrangements, and understandings between the parties regarding such subject matter, except where the parties have a signed, written master purchase agreement or similar contract. In the event of a conflict between any term of this End User License Agreement and attachments, the terms of the attachment will control solely with respect to the subject matter contained therein. Any conflicting or additional terms in your purchase orders or other documentation are expressly rejected. This Agreement may be modified only in writing, signed by authorized representatives of both parties. No use of trade or other regular practice or course of dealing between the parties will be used to modify, interpret, supplement, or alter the terms of this Agreement. No failure of either party to exercise any power or right hereunder or to insist upon strict compliance with the terms of this Agreement, and no custom or practice of the parties at variance with the terms hereof, will constitute a waiver of either party's right to demand compliance with the terms of this Agreement. If any of the provisions of this Agreement are determined to be invalid, illegal, or unenforceable, such provisions will be severed from the Agreement, and the remainder of this Agreement will be valid and enforceable to the extent permitted by applicable law, provided that the intent of the parties is not materially impaired. The parties will use their best efforts to replace the invalid or unenforceable provision by a provision that, to the extent permitted by law, achieves the purposes intended under the invalid or unenforceable provision. This Agreement will be governed by the laws of the Commonwealth of Massachusetts without regard to choice of law rules, and you hereby submit to the jurisdiction of the federal and state courts located in said Commonwealth and the applicable service of process. The parties agree that the United Nations Convention on International Sale of Goods Acts will not apply to this Agreement. Except for the obligation to make payments, non-performance of either party will be excused to the extent that performance is rendered impossible by strike, fire, flood, acts of God, governmental acts or orders or restrictions, act of terrorism, war, or any other reason where failure to perform is beyond the reasonable control of the non-performing party and not due to its fault or negligence.
- 15. High Risk Activities.** The Product is not fault-tolerant and is not designed or intended for use in hazardous environments requiring fail-safe performance, including without limitation in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems, direct life-support machines, or any other application in which the failure of the Product could lead directly to death, personal injury, or severe physical or property damage (collectively, "High Risk Activities"). Zyriion expressly disclaims any express or implied warranty of fitness for High Risk Activities.
- 16. Maintenance and Support Services.** Maintenance and technical support services ("Maintenance") for the Products will be made available in accordance with Zyriion's then-current support services terms, provided upon request.
- 17. APIs and Software Development Kits.** Zyriion may make APIs generally available. You may use the SDK to design, develop, and test software programs; make a single copy of the SDK for back-up purposes only; copy the runtime components of the SDK ("Runtime Component") into software code created through your use of the SDK; and reproduce and distribute such Runtime Component solely as a component of your software code. In addition to the restrictions in Section 2, you may not use the SDK to develop a product or service that competes with products or services offered by Zyriion, or incorporate the Runtime Component in a product that competes with the products offered by Zyriion.

Zyriion's ownership rights in Section 2 apply to the SDK, including any output such as the Runtime Component, but do not include any original software code you may develop. The inclusion of the Runtime Components in your original code

created through your use of the SDK in no way alters Zyrion's ownership rights in the Runtime Component. Zyrion may develop software programs substantially similar or identical to those developed by you through your use of the SDK and reserves the right to sell and distribute those software programs.

Zyrion does not offer Maintenance or any other support on the SDK and may change, suspend, or discontinue any aspect of the SDK at any time, including the availability of any SDK, and impose limits on certain features and services or restrict your access to parts or all of the SDK.

ZYRION DOES NOT REPRESENT OR WARRANT THAT THE SDK IS FREE OF ERRORS, BUGS, OR INTERRUPTIONS OR IS RELIABLE, ACCURATE, OR COMPLETE. ZYRION HEREBY DISCLAIMS ALL LIABILITY FOR ANY CLAIMS, DAMAGE TO, OR OTHER IMPACT ON YOUR BUSINESS, EQUIPMENT, HARDWARE, SOFTWARE, DATA, OR OTHER INFORMATION OR MATERIALS RESULTING FROM, CAUSED BY, OR RELATED TO THE USE OF THE SDK. YOUR USE OF THE SDK IS AT YOUR OWN DISCRETION AND RISK, AND YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGE THAT RESULTS FROM THE USE OF ANY SDKs. THE SDK IS PROVIDED "AS IS" WITH NO WARRANTY, EXPRESS OR IMPLIED, AND ZYRION EXPRESSLY DISCLAIMS ALL WARRANTIES AND CONDITIONS, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT.

To the maximum extent permitted by applicable law, you hereby (a) release and waive all claims against Zyrion and its subsidiaries, affiliates, officers, agents, licensors, and employees (collectively "Indemnitees") from liability for claims, damages (actual and consequential), and expenses (including litigation costs and attorneys' fees) arising from or in any way related to your use of the SDK; and (b) agree to hold harmless and indemnify Indemnitees from and against any third party claims arising from or in any way related to your use of the SDK, including any liability or expense arising from all claims, losses, damages (actual and consequential), suits, judgments, litigation costs, and attorneys' fees.

Preface

About this Guide

This reference guide describes the Application Programming Interface (API) and Web Services interface for Zyrion's Traverse.

Audience

This guide is intended for programmers who are familiar with the Traverse software and wish to extend its functionality using the provided APIs and Web services interface.

Upgrading from NetVigil

IMPORTANT: If you are upgrading from NetVigil to Traverse, you must review the namespace used in your current scripts. While full attempts have been made to keep the APIs backward compatible with NetVigil, there might be circumstances where the namespace has changed. All such incompatibilities have been marked with "NETVIGIL COMPATIBILITY" in this user guide.

Getting More Information

For more information about Zyrion's Traverse, refer to the following documents:

- *Traverse User Guide*
- *Traverse Release Notes*

Contacting Zyrion

Contacting Zyrion, Inc.

Customer Support

You can reach Zyrion technical support online:

<http://www.zyrion.com/support>

Telephone

In the US: **877-7-ZYRION (877-799-7644)**

Outside the US: **+1 408-524-7424**

Email

support@zyrion.com

User Forum

To join a customer-driven user group connecting the worldwide community of Zyrion users, visit our online forum:

<http://forums.zyrion.com/>

Contents

Preface	7
About this Guide	7
Audience	7
Upgrading from NetVigil	7
Getting More Information	7
Contacting Zyrion	8
1 BVE FlexAPI Protocol Reference	17
Overview	17
Connecting to the Server	17
Disconnecting from the Server	18
Command/Reply Formatting Rules	18
Client Command Format	18
Server Response Format	19
Client Commands	20
Login	20
Logout Quit	21
action.create	21
action.update	21
action.delete	22
action.list	22
adminGroup.x	22
adminClass.create	22
adminClass.update	23
adminClass.delete	23
adminClass.list	23
container.create	23
container.delete	27
container.list	27
container.members	28
container.status	28
container.update	28

department.create	30
department.update	31
department.delete	32
department.suspend	32
department.resume	32
department.list	32
device.create	33
device.update	34
device.delete	36
device.suspend	36
device.resume	36
device.list	36
device.status	37
device.export	37
deviceDependency.create	38
deviceDependency.delete	38
deviceDependency.list	38
dge.create	39
dge.update	39
dge.delete	39
dge.list	39
event.list	40
location.create	40
location.update	40
location.delete	41
location.list	41
result.list	41
test.create	41
test.update	45
test.delete	46
test.suspend	46
test.resume	46
test.suppress	47
test.list	47
test.status	48
user.create	48
user.update	49
user.delete	49

user.list	50
user.represent	50
userClass.create	50
userClass.update	50
userClass.delete	50
userClass.list	51
Further Examples	51
2 External Data Feed (EDF) Reference	53
Overview	53
Connecting To The Server	53
Disconnecting From the Server	54
Command/Reply Formatting and Commands	54
Client Command Format	54
Server Response Format	55
Client Commands	55
Example	56
Templates for EDF Tests	58
EDF Monitors and Plugin Monitors	59
3 Traverse Perl API	61
Overview	61
Zyriion::ExternalData - EDF API	61
Methods	61
Zyriion::Message - ISM API	64
Methods	64
Zyriion::Provisioning - BVE API	67
Methods	67
Examples	71
4 Plugin Monitors	77
Overview	77
Adding A New Test Type	77
Sample TestTypes.xml Entry	78
Creating A New Plugin Java Monitor	80
Configuration File Format	80
Writing The Plugin Class	82
Configuring the Plugin Package	83

Provisioning Plugin Tests	84
Creating A New Plugin Script Monitor	84
Configuration File Format	84
Writing The Plugin Script	87
Sample Plugin Monitor with Discrete Thresholds	87
Extending the Message Handler	89
00_src_logs.xml	89
00_src_traps.xml	90
5 External Authentication	91
Overview	91
Authentication Plugin Java Class	91
Authentication Plugin Script	93
Web Portal Authentication	95
Architectural Description	95
6 Plugin Actions	97
Overview	97
Creating New Plugin Actions	97
writeToFile.xml	97
writeToFile.sh	100
Examples	101
Reboot Router	101
Extending the Action Framework	102
RT Trouble Ticketing Plugin	103
7 Web Services API Overview	111
About the Traverse Web Services API	111
Why Use the Traverse Web Services API	112
Traverse Web Services API Workflow	112
About the Traverse Web Services	113
User Types	113
Traverse WSDL Files	113
WSDL File Locations	114
Sample Code	114
8 Session Manager Service	115

Overview	115
Session Types	115
Login Type Parameters	115
Operations	116
login	116
logout	117
Types	118
Complex Types	118
Simple Types	121
9 Business Visibility Engine Service	123
Overview	123
Operations	124
getContainerStatus	124
getDepartmentStatus	125
getDeviceStatus	125
getSubnetStatus	126
getTestResult	126
getTestStatus	126
listContainerObject	127
listDepartmentObject	127
listDeviceObject	127
listDgeObject	127
listLocationObject	128
listMonitorConfigObject	128
listSubnetDeviceObject	128
listSubnetObject	129
listTestObject	129
listUserObject	129
Types	130
10 External Data Feed Service	165
Overview	165
Operations	165
insertTestResults	165
Complex Types	166
ArrayOfBaseResponse	166
ArrayOfTestResult	166

BaseResponse	166
ResultInsertRequest	167
ResultInsertResponse	167
TestResult	168

SECTION 1

Developer API



1.1 Overview

The Business Visibility Engine (BVE) in Traverse handles the distributed architecture transparently and provides a common interface for provisioning, data and report retrieval. Through the BVE, you can access the provisioning database and real-time statistics.

The BVE server is accessed via a text based protocol over a TCP socket. Protocol messages can be sent from programs written in C, Java, Perl or any other language.

An alternative to accessing the BVE Server directly is to use the Traverse Perl API described in [Traverse Perl API on page 61](#).

1.2 Connecting to the Server

Communication with the BVE Server consists of two phases - a connection establishment phase and a command-execution phase. The connection establishment phase is where a remote client provides authentication information to the server in the form of a login id that is associated with a department, and the corresponding password. Once the authentication information has been verified, all subsequent commands sent to the server will be executed with the permissions and privileges of the specified department. It is possible to change the privilege level at any point in the command-reply phase by entering new authentication information.

Once the connection establishment phase has been completed, the client application may send one command at a time and wait to receive a reply (possibly consisting of multiple lines of output) from the server.

A client application establishes a connection to the BVE Server by opening a TCP/IP socket, using the hostname/IP of the server that is hosting the provisioning database, and a pre-defined port number specified in `dge.xml` (default port number is 7661). Upon establishment of the TCP session, the server will greet the client with a welcome message following the rules outlined below. The client may optionally send client-side information, e.g. client software version.

1.3 Disconnecting from the Server

When the client application would like to disconnect from the Traverse platform, it should issue a disconnect request instead of simply closing the socket connection. This will allow the server to perform proper cleanup before disconnecting the session.

Also if the BVE Server does not receive anything from the client for an extended period, the session will timeout and disconnect the client from the BVE Server. The default timeout is currently 15 minutes.

1.4 Command/Reply Formatting Rules

The commands sent by a client and responses sent back by the server must adhere to the following formatting conventions:

Client Command Format

- Each client command is composed of a single line of text terminated by a newline character. A carriage return followed by a newline (`\r\n`) is considered to be the same as a newline character (`\n`) alone.
- Client commands may or may not require additional parameters. Each parameter consists of the option name and value, separated by equal sign (=) and enclosed in quotes. Multiple parameters should be separated by a comma (,) and space. Example `command1=value1, command2=value2 ...`. If a parameter supports multiple values, the values should be separated by a comma (,). Example: `command1=value1, value2, ...`
- No whitespace characters should appear in the argument name. No whitespace should occur between the argument name and the equal (=) delimiter. Whitespace that occurs after the equal (=) is considered to be part of the argument value.
- Double-quotes are not permitted as part of the value.
- For each client command, the server will respond with a response code indicating success or failure, and optionally some descriptive text indicating actions taken.
- If a client command produces a reply with more than one line of output, the server will respond with 203 response code (see below). Each set of output will be terminated with a newline (`\n`) and the end of the array will be indicated with a newline (`\n`) by itself.
- Command and parameter names are NOT case sensitive.

- Parameters for any command may appear in any order following the command.
- Certain parameters indicate a value of `<regex>`, which indicates that the parameter may point to a non-unique value. You can use an asterisk (*) as a wildcard in such case. For example, `dev*` would match `device1` and `Device B` but not `my device`. However, `*dev*` would match all three.
- Parameters which indicate `<value>` require a value which is already present in the database, while `<new_value>` indicates a new value to be inserted into the database.
- All date/time should be specified in `YYYYMMDDhhmm` format and in GMT.
- For `startTime`, a blank value (or zero [0]) indicates 24 hours ago and for `endTime`, it indicates now.
- `timezone_value` is specified in format listed at <http://www.timezoneconverter.com/cgi-bin/zonehelp.tzc?cc=US>.
- For `Device`, `Test`, and `Container` specific commands, a special parameter `userName=<value>` is available to admin users. When used, the object would be created/deleted/etc. as the specified user.

Server Response Format

The server will always respond (to client initiated commands/requests) with text of the following format:

```
<status code> <response code> [optional informative text]
```

where `status code` is one of:

- `OK`: which indicates the command/request was successful
- `ERR`: which is indicative of failure to execute the request

and `response code` is a three digit numeric code which provides additional details about the `status code`.

Table 1-1. Server Response Codes

Response Code	Description
200	Server ready for initial handshake
201	Request accepted and processed, ready for next request
203	Request accepted, multi line response follows
298	Request accepted, server will halt
299	Session ended, server will close socket connection
300 - 399	Debugging information. These messages will be printed before a 200, 400 or 500 level message
400 level - client side error (try again)	
401	Authentication failure
402	Logged in user does not have permission to perform requested task
410	Unknown command - use 'help' for list of commands
411	Feature has not been implemented yet
412	Not enough parameters specified - use 'help command.name'
413	Invalid Parameter parameter specified - use 'help command.name'
420	One or more objects already exist
421	One or more of the objects requested does not exist
500 level - server side error	
595	Communication failure with remote database, please try again later
596	Maintenance in progress, please try again later
597	Server too busy, please try again later
598	Backend failure, server will close socket connection
599	Server unavailable, server will close socket connection

1.5 Client Commands

Login

Provides authentication information to the server.

```
LOGIN <login_id>/<password>
```

Logout | Quit

Ends a login session.

```
LOGOUT
```

action.create

Creates a new action. Using '0' for `n_tests` on `notifyAfter` or `repeat` parameters will send an immediate notification and will not repeat the notification respectively. It is possible to get notifications on multiple states by specifying different state names separated by '|' symbol for `notifyOn` parameter. Assigning a method of 'none' will delete that action item. Up to five methods (method1 through method5) can be defined for a single action.

```
action.create "actionName=<new_value>"
[, "method<1..5>=<none|email>"]
[, "notifyOn<1..5>=<ok|warning|critical|unknown>"]
[, "recipient<1..5>=<new_value>"]
[, "notifyAfter<1..5>=<n_tests>"]
[, "repeat<1..5>=<n_tests>"]
[, "description=<new_value>"]
```

action.update

Updates configuration information of one of more existing actions. If `actionSerial` and `actionName` are both given, then the action matching the serial number given by `actionSerial` is updated with the name given by `actionName`. Omitting an action item implies intent to remove that particular item. So if there are two action items for the action profile, and you are updating the parameters for action item #2 (method2), you must include the details of method1 verbatim (available via `action.list` command) first, and then the updated parameters for method2. Otherwise action item #1 is removed.

```
action.update <"actionName=<regex>" | "actionSerial=<value>">
, "method<1..5>=<none|email>"
, "recipient<1..5>=<new_value>"
```

```
, "notifyOn<1..5>=<ok, |warning, |critical, |unknown>"  
[, "notifyAfter<1..5>=<n_tests>"]  
[, "repeat<1..5>=<n_tests>"  
[, "description=<new_value>"]
```

action.delete

Deletes one or more existing actions.

```
action.delete <"actionName=<regexp>", | "actionSerial=<value>">  
[, "method=<regexp>"]  
[, "recipient=<regexp>"]  
[, "description=<regexp>"]
```

action.list

Lists actions based on search criteria.

```
action.list  
["actionName=<regexp>" | "actionSerial=<value>"]
```

adminGroup.x

The adminGroup commands are similar to the department.x commands below in syntax which are described below. Please see the description of these commands below.

adminClass.create

Creates an administrative group. Administrative groups are assigned User Groups, and members of admin groups can access information on any device, that is part of a department under a user group assigned to that particular admin group.

```
adminClass.create "groupName=<new_value>" [, "comment=<new_value>"]  
[, "userClasses=<new_value, ...>"]
```

adminClass.update

Updates user group assignments or name of an existing admin group. If `AdminClassSerial` and `groupName` are both given, then the admin group name will be updated.

```
adminClass.update <"groupName=<regex>" | "adminClassSerial=<value>">
[, "comment=<new_value>"]
[, "userClasses=<new_value,...>"]
```

adminClass.delete

Deletes an existing admin group.

```
adminClass.delete "groupName=<regex>" | "adminClassSerial=<value>">
```

adminClass.list

Lists admin group information based on search criteria.

```
AdminClass.list
["groupName=<regex>" | "adminClassSerial=<value>"]
```

container.create

Creates containers.

```
container.create "serviceName=<value>"
, "serviceType=<device|test>"
, "memberListMethod=<auto|manual>"
, {membership_parameters}
, {severity_parameters}
[, "parentNames=none|<value1,value2,...>"]
[, "actionName=none|<value>"]
[, "comment=<value>"]
[, "displayComment=<true|false>"]
```

Note the following:

- The container name must be unique within the end user department or admin-group, and must not be case sensitive. The container name cannot be the same as a device in your Traverse environment. Containers cannot have the same name as an existing device.
- When `serviceType=device` (default) and `memberListMethod>manual` (default), `{membership_parameters}` includes the following:


```
, "memberList=<regexp_1,regexp_2,...>"
```
- If any (comma-separated) entry for the value of `memberList` begins with #, it indicates another container that is nested below this new container. Administrator users can specify devices from end user departments by using the department name as a prefix with - (dash) as a separator.

Example:

```
container.create "serviceName=My Admin Container",
, "serviceType=device"
, "memberListMethod>manual"
, "memberList=Corporate - cisco*,#Another Admin Container"
, "severityMethod=auto"

container.create "serviceName=All End User Containers",
, "serviceType=device"
, "memberListMethod>manual"
, "memberList=* - #*"
```

Note the use of a wildcard for both the department and container names. If a matching department name is not found, use the entire entry as a device in the user's department.

- When `serviceType=device` and `memberListMethod=auto`, `{membership_parameters}` includes the following:


```
[, "ruleDeviceName=<value>"]
[, "ruleDeviceType=<value>"]
[, "ruleDeviceModel=<value>"]
[, "ruleDeviceVendor=<value>"]
[, "ruleDeviceTag1=<value>"]
[, "ruleDeviceTag2=<value>"]
```

```
[, "ruleDeviceTag3=<value>"]
[, "ruleDeviceTag4=<value>"]
[, "ruleDeviceTag5=<value>"]
```

- You must specify at least one rule. You can specify each rule parameter only once. The value supports multiple regular expressions.

Example:

```
container.create "serviceName=San Jose Devices"
, "serviceType=device"
, "memberListMethod=auto"
, "ruleDeviceName=*sjc*",
, "ruleDeviceType=unix*",
, "actionName=HQ Failure"
, "displayComment=false"
```

- When `serviceType=test`, `memberListMethod` is an optional parameter with "manual" as the only valid value. `{membership_parameters}` includes the following:

```
, "testListMethod=<auto|manual>"
[, "testName=<regexp_1,regexp_2,...>"]
[, "testType=<type_subtype_pair_1,type_subtype_pair_2,...>"]
```

- When `serviceType=test` and `testListMethod=auto`, `testName` and `testType` parameters are not required. Instead, you must include all current and future tests for the selected devices in the container.

Example:

```
container.create "serviceName=Router Tests"
, "serviceType=test"
, "memberListMethod=manual"
, "memberList=cisco*",
, "testListMethod=auto"
, "actionName=none"
```

- When `serviceType=test` and `testListMethod=manual`, the `testName` parameter is required. `testType` is an optional parameter that you can use to further filter the list of tests.

Example:

```
container.create "serviceName=All RTT Tests"  
  
  , "serviceType=test"  
  
  , "memberListMethod=manual"  
  
  , "memberList=*",  
  
  , "testListMethod=manual"  
  
  , "testName=*"   
  
  , "testType=ping/rtt"  
  
  , "actionName=Slow Response"  
  
  , "comment=Response Time to Remote Sites"  
  
  , "displayComment=true"
```

- `{severity_parameters}` includes the following, where N = 1, 2, or 3:

```
[, "severityMethod=<auto|manual>"]  
  
[, "ratioN=<value>"]  
  
[, "memberSeverityN=<ok|unknown|warning|critical>"]  
  
[, "serviceSeverityN=<ok|unknown|warning|critical>"]
```

If `severityMethod=auto`, the remaining parameters are not required. If `severityMethod=manual`, you must specify at least one set of parameters. A complete set includes all three parameters.

- You can use the `parentNames` parameter to nest the a newly created container under other existing containers. Because you can nest a container below multiple containers, the value can specify a comma-separated list of existing containers. However, you can only specify device container names. A value of `none` indicates that the container is a top level container.

Example:

```

container.create "serviceName=San Jose Devices"
, "serviceType=device"
, "memberListMethod=auto"
, "ruleDeviceName=*sjc*",
, "ruleDeviceType=unix*",
, "parentNames=Critical Servers,HQ"

```

container.delete

Deletes a container.

```

container.delete <"serviceName=<regexp>" |
"serialNumber=<value>">
[, "moveChildren=<parent|top|delete>"]

```

- If you are deleting a device container that has other containers nested below it, `parent` is used as the default value. The nested containers are moved to the immediate parent of the container you are deleting.

If `moveChildren=top`, the nested containers are moved to the top level while preserving their hierarchy. If `moveChildren=delete`, all nested containers are deleted recursively, unless a nested container is specified under a different hierarchy.

Example:

```

container.delete "serviceName=*HQ*"
, "moveChildren=delete"

```

- Administrator users can delete containers in their own admin-group. Administrators can use the `userName=<value>` parameter to delete a container in end user departments. However, this action is determined by the admin-class permission configuration.

container.list

Lists container information based on search criteria.

```

container.list <"serviceName=<regexp>" |
"serialNumber=<value>">
[, "serviceType=<device|test>"]
[, "memberListMethod=<auto|manual>"]
[, "severityMethod=<auto|manual>"]

```

```
[, "parentNames=none|<value1,value2,...>"]  
[, "actionName=none|<value>"]
```

- Use the optional parameters as search filters to narrow the listed containers. You can further filter the results by using other parameters (for example, `ruleDeviceName` when `serviceType=device` and `memberListMethod=auto`).
- The output includes all parameters from the `container.update` command, with the exception of `newServiceName` and `memberList`.

container.members

Lists the members of a container.

```
container.members <"serviceName=<regexp>" |  
"serviceSerial=<value>">
```

Output is in the following format:

```
"serviceName=<value>", "memberType=<device|container|test>",  
"memberName=<value>", "memberStatus=<severity>",  
"deviceName=<value>", "accountName=<value>",  
"deviceSerialNumber", "testSerialNumber"
```

- When `memberType=device` or `memberType=container`, `deviceName` is empty and `accountName` provides the name of the department for the device or container.
- When `memberType=test`, `deviceName` is the name of the real device and `accountName` provides the name of the department for that device.

container.status

Displays a summary of containers that have been created. The aggregate severity of each container is provided.

```
container.status  
["serialNumber=<value>" | "serviceName=<regexp>"]
```

container.update

Updates a container.

```
container.update <"serviceName=<regexp>" |  
"serialNumber=<value>">  
[, "newServiceName=<value>"]
```

```
[, "serviceType=<device|test>"]
[, "memberListMethod=<auto|manual>"]
[, "memberList=[+,]<regexp_1,regexp_2,...>"]
[, "ruleDeviceName=<value>"]
[, "ruleDeviceType=<value>"]
[, "ruleDeviceModel=<value>"]
[, "ruleDeviceVendor=<value>"]
[, "ruleDeviceTag1=<value>"]
[, "ruleDeviceTag2=<value>"]
[, "ruleDeviceTag3=<value>"]
[, "ruleDeviceTag4=<value>"]
[, "ruleDeviceTag5=<value>"]
[, "testListMethod=<auto|manual>"]
[, "testName=<regexp_1,regexp_2,...>"]
[, "testType=<type_subtype_pair_1,type_subtype_pair_2,...>"]
[, "severityMethod=<auto|manual>"]
[, "ratioN=<value>"]
[, "memberSeverityN=<ok|unknown|warning|critical>"]
[, "serviceSeverityN=<ok|unknown|warning|critical>"]
[, "parentNames=none|<value1,value2,...>"]
[, "actionName=none|<value>"]
[, "comment=<value>"]
[, "displayComment=<true|false>"]
```

- `serviceType` is a required parameter when you use the `memberList` parameter.
- `memberListMethod` is a required parameter when you use the `serviceType=device` parameter.
- You can use the `newServiceName` parameter to rename an existing service container. Container names must be unique within the end user department or admin-group, and cannot be case sensitive.

- Depending on the value of the `serviceType`, `memberListMethod`, and `testListMethod` parameters, different `{membership_parameters}` are available (same as the [container.create](#) command).
- If the value for the `memberList` parameter contains `memberListAppend` as one of the (comma-separated) entries, you must add the remaining entries to the container in addition to existing devices. Without `memberListAppend`, replace the list of devices with the new list provided.

Example:

```
container.update "serviceSerial=12345"

"memberList=new_device_1,memberListAppend,#new_container_2,router*"

```

- If you are changing a device container to a test container (`serviceType=test`) and there are nested containers below it, move those containers to the immediate parent of the container you are modifying.

Example:

DC1

```
memberListAppend=true-DC2
```

```
memberListAppend=true-DC3
```

```
memberListAppend=true-TC1
```

```
container.update "serviceName=DC3"
, "newServiceName=TC2"
, "serviceType=test"

```

This results in a new hierarchy:

DC1

```
memberListAppend=true-DC2
```

```
memberListAppend=true-TC1
```

```
memberListAppend=true-TC2
```

department.create

Creates new department information. A user login of the same name as the newly created department will also be created with the specified password.

```
department.create "departmentName=<new_value>",
"groupName=<new_value>",
"password=<new_value>",
"passwordVerify=<new_value>",
"contactEmail=<new_value>",
"contactPhone=<new_value>"
[, "company=<new_value>"]
[, "address1=<new_value>"]
[, "address2=<new_value>"]
[, "city=<new_value>"]
[, "state=<new_value>"]
[, "zip=<new_value>"]
[, "country=<new_value>"]
```

department.update

Updates information for an existing department.

```
department.update <"departmentName=<regex>" |
"departmentSerial=<value>">
[, "groupName=<new_value>"]
["company=<new_value>"]
[, "contactEmail=<new_value>"]
[, "contactPhone=<new_value>"]
[, "address1=<new_value>"]
[, "address2=<new_value>"]
[, "city=<new_value>"]
[, "state=<new_value>"]
[, "zip=<new_value>"]
[, "country=<new_value>"]
```

department.delete

Deletes an existing department. Any login IDs, devices and tests associated with this department would automatically get deleted as well.

```
department.delete <"departmentName=<regexp>" |  
"departmentSerial=<value>">
```

department.suspend

Suspends an existing department. All login IDs associated with this department would be locked out of the system and all devices/tests for the corresponding login IDs would also be suspended.

```
department.suspend <"departmentName=<regexp>" |  
"departmentSerial=<value>"> , "reason=<new_value>"
```

department.resume

Unsuspects a previously suspended department. All login IDs associated with this department would be able to log into the system once again and all devices/tests for the corresponding login IDs would start to be monitored again.

```
department.resume <"departmentName=<regexp>" |  
"departmentSerial=<value>">
```

department.list

Lists department information based on search criteria.

```
department.list  
  
["departmentName=<regexp>" | "departmentSerial=<value>"]  
[, "groupName=<regexp>"]  
[, "company=<regexp>"]  
[, "contactEmail=<regexp>"]  
[, "contactPhone=<regexp>"]  
[, "address1=<regexp>"]  
[, "address2=<regexp>"]  
[, "city=<regexp>"]  
[, "state=<regexp>"]
```

```
[,"zip=<regexp>"]
[,"country=<regexp>"]
```

device.create

Creates a new device configuration in the database.

Specify the rediscovery frequency in minutes. 720 (or 12 hours) is the minimum value accepted by the system.

When you create a new device using `Device.create`, the `rediscoveryEnabled` parameter is optional. If you do not specify this parameter, the department-specific default values (configured in the Web application in Administration > Other > Test Parameter Discovery) are used. If `rediscoveryEnabled=true`, you must configure the remaining rediscovery parameters.

```
device.create "deviceName=<new_value>",
"address=<new_value>",
"locationName=<new_value>",
"deviceType=<nt|unix|router|switch|firewall|slb|proxy|vpnc|printer|wireless|other>",
"snmpCid=<new_value>">
[,"comment=<new_value>"]
[,"parentNames=<new_value,...>"]
[,"clearOnOk=<true|false>"]
[,"smartNotify=<true|false>"]
[,"showOnSummary=<true|false>"]
[,"tag1=<string>", "tag2=<string>", ... ,"tag5=<string>"]
[,rediscoveryEnabled=<true|false>]
[,rediscoveryNewTestsAction=<logOnly|updateAndLog|ignore>]
[,rediscoveryUpdatedTestsAction=<logOnly|updateAndLog|ignore>]
[,rediscoveryDeletedTestsAction=<logOnly|updateAndLog|ignore>]
[,rediscoveryFrequency=<new_value>]
```

Example:

```
Device.create "deviceName=Cisco Router 01",
```

```
"address=206.33.183.211",  
"locationName=Princeton Dev Lab",  
"deviceType=router",  
"snmpCid=public",  
"clearOnOk=true",  
"smartNotify=true",  
"showOnSummary=true"
```

device.update

Updates configuration information for one or more existing devices. If `deviceSerial` and `deviceName` are both given, then the device name will be updated.

To change the IP address of a host, you must specify the new IP address in the `newaddress` parameter. An error will be generated if more than one host matches the search criteria while changing the IP address.

For rediscovery options, specify the rediscovery frequency in minutes. 720 (or 12 hours) is the minimum value accepted by the system.

You can use the `device.update` command to enable rediscovery for one or more devices using the `rediscoveryEnabled=true` parameter. If you do not specify action and frequency parameters, department/global defaults values are used. If you specify action and frequency parameters without the `rediscoveryEnabled` parameter, only devices that already have rediscovery enabled are affected by the action and frequency parameters.

Example:

```
device.update "deviceName=*",  
"rediscoveryUpdatedTestsAction=updateAndLog",  
"rediscoveryFrequency=100000"
```

This changes the `rediscoveryUpdatedTestsAction` and `rediscoveryFrequency` for devices that have rediscovery enabled (configured in the Web application). The other fields remain unchanged and devices with rediscovery disabled are not affected.

```
device.update <"deviceName=<regex>" | "deviceSerial=<value>">  
[, "address=<regex>"]  
[, "newaddress=<ip_addr>"]
```

```
[, "snmpCid=<new_value>"]
[, "comment=<new_value>"]
[, "deviceType=<nt|unix|router|switch|firewall|slb|proxy|vpnc|p
rinter|wireless|other>"]
[, "parentNames=<new_value,new_value,...>"]
[, "clearOnOk=<true|false>"]
[, "smartNotify=<true|false>"]
[, "showOnSummary=<true|false>"]
[, "tag1=<string>", "tag2=<string>", ... ,"tag5=<string>"]
[, rediscoveryEnabled=<true|false>]
[, rediscoveryNewTestsAction=<logOnly|updateAndLog|ignore>]
[, rediscoveryUpdatedTestsAction=<logOnly|updateAndLog|ignore>]
[, rediscoveryDeletedTestsAction=<logOnly|updateAndLog|ignore>]
[, rediscoveryFrequency=<new_value>]
```

Setting Default SNMP Query Optimization

NOTE: You can enable/disable SNMP Query Optimization when you create SNMP tests. See the Traverse User Guide for more information on the SNMP monitor and query optimization.

You can specify the default setting for SNMP Query Optimization with the `device.update` command and `snmpOptimize=<0|1>`.

When enabled, SNMP Query Optimization increases the performance and efficiency of the SNMP monitor and reduces Traverse-initiated network communications.

When disabled, the DGE stops grouping SNMP queries targeted for that device in a single packet. Each test is executed through a new UDP packet with a single SNMP GET request. This will allow Traverse to monitor older devices that are unable to process multiple queries in a single request, or devices that restrict packet sizes. Disabling SNMP Query Optimization adversely affects overall scalability and should be done when absolutely necessary.

Enter:

```
device.update "devicename=LAN Switch (1-6 Net)", "snmpOptimize=0"
```

OK 201 1 NetworkDevice(s) updated.

1 enables optimization and 0 disables optimization. device.list output indicates the setting:

```
device.list "devicename=LAN Switch (1-6 Net)"
```

OK 203 request accepted, records returned: 1

```
"serialNumber=270019", "deviceName=LAN Switch (1-6 Net)", "address=10.1.6.1",  
"snmpCid=public", "snmpPort=161", "snmpVersion=2", "snmpOptimize=0", [...]
```

device.delete

Deletes configuration information for one or more devices. All associated tests for the devices are automatically deleted as well.

```
device.delete<"deviceName=<regex>" | "deviceSerial=<value>">  
  
[, "address=<regex>"]  
  
[, "locationName=<regex>"]  
  
[, "snmpCid=<regex>"]  
  
[, "deviceType=<regex>"]
```

device.suspend

Suspends one or more existing devices and all corresponding tests.

```
device.suspend <"deviceName=<regex>" | "deviceSerial=<value>">
```

device.resume

Resumes one or more previously suspended devices and all corresponding tests.

```
device.resume <"deviceName=<regex>" | "deviceSerial=<value>">
```

device.list

Lists device information based on a search criterion. Using multiple search criteria is not supported.

```
device.list  
  
["deviceName=<regex>" | "deviceSerial=<value>"]  
  
[, "tagN=<regex>" ]
```

```
[, "parentNames=<value1,value2,...>" ]
```

The output of `Device.list` shows the current rediscovery setting.

Example:

```
"serialNumber=520003", "deviceName=my_device_1",
"address=172.21.8.25", [...] "rediscoveryEnabled=true",
"rediscoveryNewTestsAction=logOnly",
"rediscoveryUpdatedTestsAction=logOnly",
"rediscoveryFrequency=720",
"rediscoveryDeletedTestsAction=logOnly"
```

device.status

Displays summary of devices being monitored. Tests for each device are displayed in the same three-column manner as in the Web application.

```
device.status ["deviceName=<regex>" | "deviceSerial=<value>"]
[, "status=<ok|warning|critical|unknown|unreachable>" ]
[, "parentNames=<value1,value2,...>" ]
```

device.export

Exports device views from one department to another. In order to export a device, you must have read/write permissions on both the source and target departments.

```
device.export
    "deviceName=<name|regex>"
    [, "newDeviceName=<name>"]
    [, "testName=<regex>"]
    , "accountName=<value>"
    , "newAccountName=<regex>"
```

Example:

```
device.export "devicename=*Cisco*", "testname=*Phone*",
"accountname=Zyrion", "newaccountname=Another Company"
```

```
OK 201 3 device(s) imported
```

If a device with same name already exists in target department, and it was previously exported from the same source department, Traverse updates the list of exported tests. Otherwise, Traverse creates the device in target department with a unique name in the following format:

```
<devicename>_imp_<timestamp>
```

deviceDependency.create

Assigns one or more existing devices as a parent device for an existing device.

```
deviceDependency.create <"deviceName=<value>","parentNames=<value,value,...>"
```

deviceDependency.delete

Deletes previously created device dependencies for one or more existing devices.

```
deviceDependency.delete <"deviceName=<regexp>","parentNames=<value,value,...>"
```

Example:

```
deviceDependency.delete "deviceName=*vlon*","parentNames=ppar2137"
```

This command is expecting devices that have a name matching `*vlon*` and have parent `ppar2137` only. If the devices have multiple parents, then you need to specify each of them using `parentNames` parameter. It can be checked using `deviceDependency.list "devicename=*vlon*"` command.

```
deviceDependency.delete "devicename=*vlon*","parentNames=ROUTERB, *"
```

This will not be successful since the command does not support wildcard for `parentNames` parameter. One option would be to write a Perl script that uses the `Zyrion::Provisioning` module and uses `ListDependency()` method to collect existing dependency information for the devices in question. Then, use that information to call `DeleteDependency()` and `CreateDependency()` methods in succession.

deviceDependency.list

Lists device dependency information based on search criteria.

```
deviceDependency.list
```

```
["deviceName=<regexp>" | "deviceSerial=<value>"]
```

dge.create

Creates a new Data Gathering Engine (DGE) instance.

```
dge.create "dgeName=<new_value>",  
"host=<new_value>",  
"locationName=<new_value>",  
"softLimit=<new_value>",  
"hardLimit=<new_value>"
```

dge.update

Updates information for an existing DGE. If `dgeSerial` and `dgeName` are both given, the DGE name is updated.

```
dge.update <"dgeName=<regex>" | "dgeSerial=<value>">  
[, "host=<new_value>"]  
[, "locationName=<new_value>"]  
[, "softLimit=<new_value>"]  
[, "hardLimit=<new_value>"]
```

dge.delete

Deletes configuration information for one or more existing DGE instances.

```
dge.delete <"dgeName=<regex>" | "dgeSerial=<value>">
```

dge.list

Lists DGE information based on search criteria.

```
dge.list  
["dgeName=<regex>" | "dgeSerial=<value>"]
```

event.list

Lists events for one or more devices and one or more tests configured on those devices for a certain time frame. By specifying a certain type of event in `eventType` parameter, it is possible to display only events where the previous or current state was of that type.

Output is in the following format:

```
device_name | device_serial_number | test_name |
test_serial_number | test_type | test_sub_type | time_stamp |
event_duration | previous_state | new_state
```

```
event.list
["deviceName=<regex>"]

[, "testName=<regex>" | "testSerial=<value>"]

[, "startTime=<YYYYMMDDhhmm>"]

[, "endTime=<YYYYMMDDhhmm>"]

[, "eventType=<ok|warning|critical|unreachable|unknown>"]

[, "testType=<regex>"]

[, "subType=<regex>"]
```

location.create

Creates a new location where one or more DGEs will be operating.

```
location.create "locationName=<new_value>",
"streetAddress=<new_value>",
"city=<new_value>",
"state=<new_value>",
"comments=<new_value>"
```

location.update

Updates information on an existing location. If both `locationName` and `locationSerial` are given, the location name is updated.

```
location.update <"locationName=<regex>" |
"locationSerial=<value>">

[, "streetAddress=<new_value>"]
```

```
[,"city=<new_value>"]
[,"state=<new_value>"]
[,"comments=<new_value>"]
```

location.delete

Deletes an existing location. All DGEs at that location and all associated devices/tests on those DGEs are deleted automatically.

```
location.delete <"locationName=<value>" |
"locationSerial=<value>">
```

location.list

Lists location information based on search criteria.

```
location.list
["locationName=<regexp>" | "locationSerial=<value>"]
```

result.list

Lists test results for one or more devices and one or more tests configured for those devices for a certain time frame.

```
result.list
["deviceName=<regexp>" | "deviceSerial=<value>"],
["testName=<regexp>" | "testSerial=<value>"]
[,"startTime=<YYYYMMDDhhmm>"]
[,"endTime=<YYYYMMDDhhmm>"]
[,"testType=<regexp>"]
[,"subType=<regexp>"]
```

Output is in the following format:

```
device_name | device_serial_number | test_name |
test_serial_number | test_type | test_sub_type | time_stamp |
num_samples | avg_value | min_value | max_value | current_state
| warning_threshold | critical_threshold
```

test.create

Creates new tests for an existing device.

Creating a ping test

```
test.create "deviceName=<new_value>",  
"testType=ping",  
"subType=pl|rtt",  
"testName=<new_value>"  
[, "interval=<new_value>"]  
[, "warningThreshold=<new_value>"]  
[, "criticalThreshold=<new_value>"]  
[, "actionName=<new_value>"]
```

Example:

```
test.create "deviceName=Cisco Router 01",  
"testType=ping",  
"subType=rtt",  
"testName=Cisco-Router-01-ping-rtt",  
"warningThreshold=250",  
"criticalThreshold=1500",  
"actionName=email-NOC"
```

Creating an SNMP test

```
test.create "deviceName=<new_value>",
"testType=snmp",
"subType=<new_value>",
"testName=<new_value>"
[, "interval=<new_value>"]
[, "warningThreshold=<new_value>"]
[, "criticalThreshold=<new_value>"]
"snmpOid=<new_value>",
"resultMultiplier=<new_value>",
"resultProcessDirective=<new_value>",
"maxValue=<new_value>",
[, "actionName=<new_value>"]
```

resultProcessDirective

Indicates what type of calculation to perform after polling the new value. For example, when it is set to “percent”, the polled value and maximum (configured) value are used to calculate percentage, which is the final result.

Table 1-2. resultProcessDirective Values

Value	Description
0	NONE. No post processing is done on the result
1	PERCENT. The fetched value is divided by the provisioned maximum value to get the percent. Useful for disk utilization.
2	DELTA. Calculate the difference between the value retrieved in the previous test and the value retrieved in the current test.
3	RATE. Calculate the delta from the previous value and then divide by the time interval between the tests to calculate the rate per second.
4	DELTAPERCENT. Calculate the delta from the previous value, and then divide by the provisioned maximum value.
5	RATEPERCENT. Calculate the delta from the previous value, and then divide by the time interval between tests as well as the provisioned maximum value to get the percentage per second.

Table 1-2. resultProcessDirective Values

Value	Description
6	REVPERCENT. Calculate the 'percent' and then subtract from 100 to get the 'reverse'. Useful to convert disk full into disk free.
7	STRHEX2LONG. Convert opaque hexadecimal strings to long (e.g. in Amperion BPC equipment)

resultMultiplier

Allows you to modify the polled result. If the SNMP agent reports data in bytes, and you want use bits, set the resultMultiplier to 8. To convert KB into MB, the resultMultiplier will be 0.001. As another example, you can also multiply by 60 to convert rate from per second to per minute.

Traverse only supports integer values for polled results, so the results are rounded off before they are stored in the database. You can use resultMultiplier to bypass this restriction. For example, if you need to monitor values up to two significant digits for load average, modify the test and enter 100 as the resultMultiplier value. You would need to update the thresholds accordingly (i.e. multiply them by 100). Note that resultMultiplier is only applicable for SNMP and "external" tests.

maxValue

This is the maximum post-processed value (not the max of the SNMP counter/gauge which is typically 2^{32}). So, if you are measuring the traffic rate of an ethernet port, which has a test unit of Kbps, the max value should be 10,000. When measuring disk space utilization, it holds the maximum size for the disk (partition) as reported by the SNMP agent, which will be used for percentage calculation.

Creating a port test

```
test.create "deviceName=<new_value>",
"testType=port",
"subType=<http|https|smtp|pop3|pop3s|imap|imaps|nntp|ftp|advanced>", "testName=<new_value>"

[, "interval=<new_value>"]

[, "warningThreshold=<new_value>"]

[, "criticalThreshold=<new_value>"]
```

```
[, "port=<new_value>"]
[, "url=<new_value>"]
[, "loginName=<new_value>"]
[, "password=<new_value>"]
[, "actionName=<new_value>"]
[, "sendString=<new_value>"]
[, "expectString=<new_value>"]
```

Creating an external test

```
test.create "deviceName=<new_value>",
"testType=external",
"testName=<new_value>",
"interval=<new_value>",
"warningThreshold=<new_value>",
"criticalThreshold=<new_value>"
[, "actionName=<new_value>"]
```

test.update

Updates configuration information for one or more existing tests.

```
test.update <"testName=<regex>">,
"deviceName=<regex>",
"testType=<value>"
[, "subType=<regex>"]
[, "interval=<new_value>"]
[, "warningThreshold=<new_value>"]
[, "criticalThreshold=<new_value>"]
[, "actionName=<new_value>"]
[, "maxValue=<new_value>"]
[, "units=<new_value>"]
```

To do a bulk update of the warning and critical thresholds for all routers named `*router*`, use the following command:

```
login localuser/localpassword

test.update "devicename=*router*", "testname=Round Trip*",
"testtype=ping", "subtype=rtt", "warningthreshold=150",
"criticalthreshold=250"
```

test.delete

Deletes configuration information for one or more existing tests. If a test name is given, then a device name is required.

```
test.delete "deviceName=<regex>",
<"testName=<regex>" | "testSerial=<value>">
[, "testType=<regex>"]
[, "subType=<regex>"]
```

NOTE: If a test is part of a Composite Test, the BVE API does not delete the test. See the *Traverse User Guide* for more information about Composite Tests.

test.suspend

Suspends testing of one or more existing tests.

```
test.suspend

<"deviceName=<regex>",
<"testName=<regex>" | "testSerial=<value>">
[, "testType=<regex>"]
[, "subType=<regex>"]
```

test.resume

Resumes regular testing for one or more previously suspended tests.

```
test.resume

<"deviceName=<regex>" | "deviceSerial=<value>">,
<"testName=<regex>" | "testSerial=<value>">
```

```
[, "testType=<regex>"]
[, "subType=<regex>"]
```

test.suppress

Suppresses the test result of one or more tests. When suppressed, the severity/state of the test will not affect the status displayed for the device/department. When test severity changes (e.g. from Warning to Critical or from Unknown to Unreachable), the suppression is reset automatically.

```
test.suppress
<"deviceName=<regex>",
<"testName=<regex>" | "testSerial=<value>">
[, "testType=<regex>"]
[, "subType=<regex>"]
```

test.list

Displays test configuration parameters for tests matching search criteria.

```
test.list
["deviceName=<regex>"]
[, "testName=<regex>" | "testSerial=<value>"]
[, "testType=<regex>"]
[, "subType=<regex>"]
```

Sample output (output is slightly test dependent):

```
"serialNumber=40003", "testName=Disk /boot Space Util",
"testType=snmp", "subType=disk", "deviceName=localhost",
"interval=300", "warningThreshold=75", "criticalThreshold=90",
"shadowWarningThreshold=75", "shadowCriticalThreshold=90",
"slaThreshold=75", "actionName=None", "suppressed=false",
"isSuspended=false", "resultProcessDirective=1", "resultMultiplier=1.0", "maxValue=101089", "snmpOid=.1.3.6.1.2.1.25.2.3.1.6.2"
```

Example:

To get test names for a device use:

```
test.list "deviceName=xyz", "testName=*"
```

test.status

Displays current status of the tests for the device specified. The search can be restricted to test names with certain pattern, or severity.

```
test.status "deviceName=<value>"  
[, "testName=<regex>" | "testSerial=<value>"]  
["status=<ok|warning|critical|unknown|unreachable>"]
```

Output is in the following format:

```
test_serial_number | current_state | avg_value |  
warning_threshold | critical_threshold | time_stamp |  
time_in_state | test_name
```

where the `time_stamp` and `time_in_state` are provided in `YYYYMMDDhhmmss` format. Note that the test name is displayed in the last field, and test serial number is in the first field.

user.create

Creates a new user (login id) in a specific department.

```
user.create "role=<read-only | read-write>",  
"loginName=<new_value>",  
"firstName=<new_value>",  
"lastName=<new_value>",  
"emailAddress=<new_value>",  
"departmentName=<new_value>",  
"password=<new_value>",  
"passwordVerify=<new_value>"  
"phoneDay=<new_value>"  
[, "phoneEvening=<new_value>"]  
[, "phoneMobile=<new_value>"]  
[, "pager=<new_value>"]  
[, "timeZone=<timezone_value>"]
```

Example:

```
user.create "role=read-only",
```

```

"loginName=jsmith",
"firstName=John",
"lastName=Smith",
"emailAddress=jsmith@acme.com",
"departmentName=roUsers",
"password=h4ckth1s!",
"passwordVerify=h4ckth1s!",
"phoneDay=609-555-1212"

```

user.update

Updates information for an existing user/login id. User login names cannot be updated, so if both `loginSerial` and `loginName` are given, the `loginName` is ignored.

```

user.update <"loginName=<regex>" | "loginSerial=<value>">
[, "role=<read-only | read-write>"]
[, "firstName=<new_value>"]
[, "lastName=<new_value>"]
[, "emailAddress=<new_value>"]
[, "departmentName=<new_value>"]
[, "password=<new_value>"]
[, "passwordVerify=<new_value>"]
[, "phoneDay=<new_value>"]
[, "phoneEvening=<new_value>"]
[, "phoneMobile=<new_value>"]
[, "pager=<new_value>"]
[, "timeZone=<new_value>"]

```

user.delete

Deletes a user/login id from a specific department.

```

user.delete <"loginName=<regex>" | "loginSerial=<value>">

```

user.list

Lists user information based on search criteria.

```
user.list  
  
["loginName=<regexp>" | "loginSerial=<value>"]  
[, "departmentName=<regexp>"]  
[, "firstName=<regexp>"]  
[, "lastName=<regexp>"]
```

user.represent

Masquerades as a specific user. This command is only available to admin users. Once executed, the permissions and privileges of the specified user will be inherited and any new department, device and tests created will be created on behalf of the specified user.

```
user.represent "loginName=<value>"
```

userClass.create

Creates a user group.

```
userClass.create "groupName=<new_value>"  
  
[, "comment=<new_value>"]
```

userClass.update

Updates user group information. If both `groupName` and `userClassSerial` are given, then the user group name will be updated with `groupName`.

```
userClass.update "<groupName=<regexp>" | "userClassSerial=<value>">  
[, "comment=<new_value>"]
```

userClass.delete

Deletes an existing user group.

```
userClass.delete "groupName=<regexp>" | "userClassSerial=<value>">
```

userClass.list

Lists user group information based on search criteria.

```
userClass.list
```

```
["groupName=<regexp>" | "userClassSerial=<value>"]
```

1.6 Further Examples

Example of a telnet session where a new device and a test are created by directly connecting to the BVE Server on TCP port 7661:

```
% telnet bve_host 7661
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
OK 200 Traverse BVE TCP Server v3.5 ready
LOGIN localuser/localpassword
OK 201 request accepted and processed, ready for next request
DEVICE.CREATE "devicename=my_device",
"address=192.168.123.25", "devicetype=unix", "snmpcid=public",
"comment=my workstation", "locationName=Denver Office"
OK 201 request accepted and processed, ready for next request
TEST.CREATE "devicename=my_device", "testname=my_test",
"testtype=external", "subtype=external", "interval=15m",
"units=xyz", "warningThreshold=55", "criticalThreshold=85",
"maxvalue=100", "resultProcessDirective=0",
"resultMultiplier=1"
OK 201 request accepted and processed, ready for next request
[[ if you wanted to check the newly created test ]]
TEST.LIST "testname=my_test", "devicename=my_device"
OK 203 request accepted, records returned: 1
"serialNumber=470003", "deviceName=my_device",
"testName=my_test", "testType=external", "subType=external",
"interval=900", "warningThreshold=55", "criticalThreshold=85",
"actionName=None", "suppressed=false", "isSuspended=false",
"resultProcessDirective=0", "resultMultiplier=1.0",
"maxValue=100"
```

QUIT

OK 299 Logging out.

Creating an advanced port test

```
test.create "devicename=test_device", "testname=SSH Service",  
"testtype=port", "subtype=advanced", "port=22",  
"expectstring=SSH", "interval=180", "warningthreshold=2",  
"criticalthreshold=5",
```

```
test.update "devicename=test_device", "testname=SSH Service",  
"testtype=port", "port=8022", "sendstring=foo",  
"expectstring=bar", "warningthreshold=3"
```

2.1 Overview

The External Data Feed (EDF) allows external data to be sent to and processed by Traverse as though it had been collected by Traverse itself. Any external tool can send results and events for any existing test, and the result/event will be processed as if a Traverse monitor had polled the result.

The EDF process is accessed via a text based protocol over a TCP socket. Protocol messages can be sent from programs written in C, Java, Perl or any other language.

Typically, you should provision the test with a type of 'external' (using the Web interface or the BVE Server) before inserting test results via the EDF server, but you can also use this process to enter data for any existing test using the test's serial number.

It is recommended that the Traverse Perl API described in [Traverse Perl API on page 61](#) be used to access the EDF instead of a direct telnet connection for consistency.

2.2 Connecting To The Server

Communication with the EDF server consists of two phases - a connection establishment phase and a command-execution phase. The connection establishment phase is where remote client provides authentication information to the server in the form of a login id, and the corresponding password. Once the authentication information has been verified, all subsequent commands sent to the server will be executed with the permissions and privileges of the specified user.

Note that the login information provided to the EDF Server is the username and password specified in the **dge.xml** configuration file and not the Web user login and password. On login, the user can insert data for all the devices and tests in Traverse.

Once the connection establishment phase has been completed, the client application may send one command at a time and wait to receive a reply (possibly consisting of multiple lines of output) from the server.

A client application establishes a connection to the EDF Server by connecting to a TCP/IP socket, using the hostname/IP of the server that is running the monitor, and a pre-defined port number (default port number is 7657). Upon establishment of the TCP session, the server will greet the client with a welcome message following the rules outlined below. If the server is ready to accept data, it will respond with

```
OK Traverse External Data Feed Server Ready
```

at which point the remote client can send authentication information. If the server is unavailable, an error message would be printed in the form

```
ERR [reason]
```

and the server will disconnect the client.

2.3 Disconnecting From the Server

When the client application would like to disconnect from the EDF Server, it is recommended that the client issue a disconnect request (see below for appropriate command) instead of simply closing the socket connection. This will allow the server to perform proper cleanup before disconnecting the session.

Also if the EDF Server does not receive anything from the client for an extended period of time, the session will timeout and disconnect the client. The default timeout is currently 2 minutes and can be changed by editing dge.xml.

2.4 Command/Reply Formatting and Commands

The commands sent by a client and responses sent back by the server must adhere to the following formatting conventions:

Client Command Format

- Each client command is composed of a single line of text terminated by a newline character. A carriage return followed by a newline (`\r\n`) is considered to be the same as a newline character (`\n`) alone.
- Client commands may or may not require additional parameters. Each parameter consists of values, separated by 'pipe' symbol (`|`). Example `command_name value1 [| value2 | value3 ..]`.

- Pipe symbol (|) is not permitted as part of the value.
- For each client command, the server will respond with a response code indicating success or failure, and optionally some descriptive text indicating actions taken.
- Command names are not case sensitive.
- Parameters/values for any command must appear in exact order following the command. If a value is not applicable or existent for a particular command, an empty value (|) should be provided

Server Response Format

The server always responds (to client initiated commands/requests) with text of the following format:

```
<status code> [optional informative text]
```

where `status code` is one of:

- OK: which indicates the command/request was successful.
- ERR: which is indicative of failure to execute the request.

Client Commands

Login

Provide authentication information to the server. This username and password are specified in the `dge.xml` configuration file.

```
Login <login_id> | <password>
```

Logout | Quit

End a login session.

```
Logout
```

Result.insert

Insert a result value for an existing test into the database.

```
Result.insert device_name | device_addr | test_name |
test_serial | date_time | result_value
```

where

- `device_name` is the descriptive name that was used when the device was provisioned.
- `device_ip` is the fully qualified address or ip address that was used when provisioning the device.
- `test_name`, along with `device_name` and `device_ip` are used to obtain the unique serial number for the test if `test_serial` is not provided. This is the descriptive test name that was used during provisioning.
- `test_serial` is the unique serial number of the test, which should be already provisioned. If no serial number is provided, the device name, address and test name (if provided) will be used to obtain the test serial number. If no test matching the serial number can be found, the result value will be ignored.
- `date_time` is provided either in `yyyy.mm.dd-hh:mm`, or `nnnnnnnnnnn` format where `nnnnnnnnnnn` is number of seconds since 1970. If the date and time are not provided, or a value of 0 is used, current system time in GMT will be used. Because of the real-time aggregation, you must provide a timestamp newer than the last data value for the test.
- `result_value` is the value which should be inserted into the database. The provided result will be multiplied by the result multiplier, and processed in the manner set via process-directive, both set during the creation of the test.

Example

- 1 The device and test need to be created in Traverse using either the Web Interface (under the Advanced Tests section) or the BVE Server. The testType should be set to `external` for EDF tests.

```
% telnet bve_host 7661
OK 200 Traverse BVE TCP Server v3.5 ready
LOGIN localuser/localpassword
OK 201 request accepted and processed, ready for next
request
```

```
DEVICE.CREATE "devicename=my_device",
"address=192.168.123.25", "devicetype=unix",
"snmpcid=public", "comment=my workstation",
"locationName=Denver Office"

OK 201 request accepted and processed, ready for next
request

TEST.CREATE "devicename=my_device", "testname=my_test",
"testtype=external", "subtype=external", "interval=15m",
"units=xyz", "warningThreshold=55",
"criticalThreshold=85", "maxvalue=100",
"resultProcessDirective=0", "resultMultiplier=1"

OK 201 request accepted and processed, ready for next
request

QUIT
```

- 2 Now connect to the EDF server on port 7657 (using the username and password in the dge.xml configuration file, which is different from the one we used to access the BVE Server in the first step. Note that we are not using the test serial number and are also specifying the timestamp as 0 which indicates use current date and time.

```
% telnet dge_host 7657

OK Traverse External Data Feed Server Ready

login edfuser|fixme

OK

result.insert my_device | 192.168.123.25 | my_test | | 0 |
25

OK

QUIT

OK Received logout - bye
```

NOTE: To view the newly inserted test result via BVE Server:

```
% telnet bve_host 7661

OK 200 Traverse BVE TCP Server v3.5 ready

login localuser/localpassword

OK 201 request accepted and processed, ready for next
request
```

```
RESULT.SEARCH "devicename=my_device", "testname=my_test",
"starttime=NOW"

OK 203 request accepted, records returned: 1

my_device|470000|my_test|470003|external|external|2003050
6100124|1|25|25|25|Ok|55|85

QUIT

OK 299 Logging out.
```

2.5 Templates for EDF Tests

You can set up templates for EDF tests. If you create an XML configuration file under the “plugin/monitors” directory (e.g. “my_edf_test.xml”) and restart the Web application and DGE components, you will see the defined tests under Administration >devices >tests >advanced tests (under external tests section). You can create additional tests with other names with same sub-type.

```
<monitor type="external">

<testtype>

  <displayName>Sample EDF test</displayName>

  <displayCategory>application</displayCategory>

  <subType>edf_1</subType>

  [...]

</testtype>

</monitor>
```

Note that the monitor type is set to external.

2.6 EDF Monitors and Plugin Monitors

Tests from a plugin monitor are executed at the specified interval by the DGE. In contrast, the DGE does not perform any tasks for EDF tests. The DGE expects to receive test results from an external data source (script, application) at specific intervals via a TCP socket. The connecting application will need to following the EDF API protocol to communicate with the DGE. The EDF monitor is useful when the metric to be monitored is on a different host that is not accessible from the DGE via standard (SNMP, WMI) or proprietary (IP based) methods. The EDF API is also scalable to a larger extent compared to plugin monitors since the remote host can insert results for multiple tests over a single TCP session.

3.1 Overview

The Traverse Perl API provides a powerful interface to the BVE, EDF and ISM servers. This API can be used to interface with other existing provisioning systems, custom monitors, etc. without worrying about the underlying connection and other protocols.

3.2 Zyrion::ExternalData - EDF API

This is a Perl module that provides an interface into Traverse monitor/message framework using the External Data Feed (EDF) API.

`Zyrion::ExternalData` uses `Zyrion::SimpleServerClient` to connect to a remote host running the External Data Feed component of Traverse. Once connected, you can log into the server and insert results for previously provisioned tests into the database.

Methods

new

Create a new `Zyrion::ExternalData` object

```
use Zyrion qw(ExternalData);

$obj = new Zyrion::ExternalData(
    [Host      => $host_name_or_ip,]
    [Port      => $tcp_port,]
    [User      => $login_id,]
    [Password => $login_password,]
    [DEBUG     => <0|1>];
```

This is the constructor for `Zyrion::ExternalData` objects. A new object is returned on success. The `$login_id` and `$login_password` parameters can be omitted and specified during `Login` method. No connection is made to the remote host when this method is called. Only the object is created with remote host address and port information. The new object returned is given the following defaults in the absence of corresponding named arguments:

- The default `Host` is `localhost`.
- The default `Port` is `7657`.

GetErrorMsg

Retrieve error information from last operation

```
$error = $obj->GetErrorMsg
```

If any previous methods have failed, this method will return relevant information, if available.

Login

Log into Traverse EDF server.

```
$return_value = $obj->Login(  
    [User      => $login_id,]  
    [Password => $login_password],  
    [Timeout  => $timeout_secs]);
```

This method opens a TCP connection to `$tcp_port` on `$host_name_or_ip`, as defined using the `new` method. If either the `$login_id` argument or the `$login_pass` argument is missing, the values specified in the `new` method (if any) are used. The username and password for the EDF server is different from the users configured into the provisioning server. A special EDF user, specific to each DGE, is configured via the `etc/dge.xml` configuration file.

An optional named argument is provided to override the current timeout setting. On timeout or other connection errors, the return value is '0' and details on the error are available via the `GetErrorMsg` method. On success, a non-zero return value is provided.

Logout

Log out of the Traverse EDF server.

```
$return_value = $obj->Logout;
```

This method sends a logout command to the Traverse EDF server and closes the already established TCP connection to `$host_name_or_ip`, which was defined using the new method.

On timeout or other connection errors, the return value is '0' and details on the error are available via the `GetErrorMsg` method. On success a non-zero return value is provided.

InsertResult

```
$return_value = $obj->InsertResult(
    [deviceName => $device_name,]
    [deviceAddr => $device_fqdn_or_ip,]
    [testName => $test_name,]
    [testSerial => $test_serial_num,]
    [dateTime   => time(),]
    [result     => $value_to_insert]
```

Refer to [External Data Feed \(EDF\) Reference on page 53](#) for explanations of the parameters and valid values.

The following example creates a connection to localhost (default port), logs in, inserts result for a test named `sample_test` into the message database, and quits:

Example: Connect, log in, insert test result, log out

```
use Zyrion qw(ExternalData);
my $obj = new Zyrion::ExternalData(Host=>"localhost");
my $return_value = $obj->
    Login(User=>"edfuser",Password=>"fixme") ||
    die "ERROR: ", $obj->GetErrorMsg, "\n";
$obj->InsertResult(deviceName=>"my_server",
    deviceAddr=>"192.168.200.50",
    dateTime=>time,
    testName=>'sample_test',
    result=>100) ||
    print "ERROR: ", $a->GetErrorMsg, "\n";
```

```
$obj->Logout;
```

Note that the optional parameter `testSerial` was not provided. So the server will use `deviceName`, `deviceAddr` and `testName` to determine test serial number.

3.3 Zyrion::Message - ISM API

This is a Perl module that provides an interface into the Traverse monitor/message framework using the Input Stream Monitor (ISM) API.

`Zyrion::Message` uses `Zyrion::SimpleServerClient` to connect to a remote host running the Input Stream Monitor component of Traverse. Once connected, you can log into the server and insert free-form or formatted messages into the database.

Methods

new

Create a new `Zyrion::Message` object

```
use Zyrion qw(Message);

$objj = new Zyrion::Message(
    [Host      => $host_name_or_ip,]
    [Port      => $tcp_port,]
    [User      => $login_id,]
    [Password => $login_password,]
    [DEBUG    => <0|1>];
```

This is the constructor for `Zyrion::Message` objects. A new object is returned on success. The `$login_id` and `$login_password` parameters can be omitted and specified during the `Login` method. No connection is made to the remote host when this method is called. Only the object is created with remote host address and port information. The new object returned is given the following defaults in the absence of corresponding named arguments:

- The default `Host` is `localhost`
- The default `Port` is `7659`

GetErrorMsg

Retrieve error information from last operation

```
$error = $obj->GetErrorMsg
```

If any previous methods have failed, this method will return relevant information, if available.

Login

Log into Traverse ISM server.

```
$return_value = $obj->Login(
    [User      => $login_id,]
    [Password => $login_password],
    [Timeout  => $timeout_secs]);
```

This method opens a TCP connection to *\$tcp_port* on *\$host_name_or_ip*, as defined using the *new* method. If either the *\$login_id* argument or the *\$login_pass* argument is missing, the values specified in the *new* method (if any) are used. The username and password for the ISM server is different from the users configured into the provisioning server. A special ISM user, specific to each DGE, is configured via the *etc/dge.xml* configuration file.

An optional named argument is provided to override the current timeout setting. On timeout or other connection errors, the return value is '0' and details on the error are available via the *GetErrorMsg* method. On success, a non-zero return value is provided.

Logout

Log out of the Traverse ISM server.

```
$return_value = $obj->Logout;
```

This method sends a logout command to the Traverse ISM server and closes the already established TCP connection to *\$host_name_or_ip*, which was defined using the *new* method.

On timeout or other connection errors, the return value is '0' and details on the error are available via the *GetErrorMsg* method. On success a non-zero return value is provided.

InsertMessage

```

$return_value = $obj->InsertMessage(
    [deviceName => $device_name,]
    [deviceAddr => $device_fqdn_or_ip,]
    [testName= => $test_name,]
    [testSerial => $test_serial_num,]
    [dateTime => time(),]
    [severity => <"ok"|"warning"|"critical">,]
    [message => $text_to_insert]);

or,

$return_value = $obj->InsertMessage("free-form text to insert");

```

The first form of the method will insert a message specific for the device (and optionally test) specified into the system for further processing. The second method will force the system to match the message against all configured regular expression patterns and if there is a match, appropriate severity will be set and actions will be triggered. See the the “Message Handler for Traps and Logs” chapter in the *Traverse User Guide* for explanations of the parameters and valid values for the first method.

The following example creates a connection to localhost, logs in, inserts a message into the message database, and quits:

Example: Connecting, Logging In, Inserting Message, Logging Out

```

use Zyrion qw(Message);

my $obj = new Zyrion::Message(Host=>"localhost");

my $return_value = $obj->
    Login(User=>"ismuser",Password=>"fixme") ||
    die "ERROR: ", $obj->GetErrorMsg, "\n";

$obj->InsertMessage(deviceName=>"my_server",
    deviceAddr=>"192.168.200.50",
    dateTime=>time,
    severity=>'ok',
    Message=>"this is a message") ||

```

```
print "ERROR: ", $a->GetErrorMsg, "\n";

$obj->Logout;
```

Note that the optional parameters `testName` and `testSerial` were not provided. So the server will use `deviceName` and `deviceAddr` to determine device serial number.

3.4 Zyrion::Provisioning - BVE API

This is a Perl module that provides an interface into Traverse's Business Visibility Engine (BVE) and provisioning database using Traverse Query Protocol (NQP).

`Zyrion::Provisioning` uses `Zyrion::SocketIO` to connect to a remote host running the Traverse BVE socket server. Once connected, you must log in as a user and then can perform create/delete/update etc. tasks on all the Traverse objects (user, device, test, etc) as well as get realtime test details and reports.

The detailed list of commands and parameters expected by the BVE socket server is detailed in [BVE FlexAPI Protocol Reference on page 17](#).

Methods

new

Create a new `Zyrion::Provisioning` object

```
$obj = new Zyrion::Provisioning(
    [Host      => $host_name_or_ip,]
    [Port      => $tcp_port,]
    [User      => $login_id,]
    [Password => $login_password,]
    [DEBUG     => <0|1>];
```

This is the constructor for `Zyrion::Provisioning` objects. A new object is returned on success. The `$login_id` and `$login_password` parameters can be omitted and specified during `Login` method. No connection is made to the remote host when this method is called. Only the object is created with remote host address and port information. The new object returned is given the following defaults in the absence of corresponding named arguments:

- The default Host is localhost.
- The default Port is 7661.

GetErrorMsg

Retrieve error information from last operation.

```
$error = $obj->GetErrorMsg
```

If any previous methods have failed, this method will return relevant information, if available.

GetResultCount

Return the number of objects in the result buffer.

```
$result_count = $obj->GetResultCount;
```

This method provides a count of the number of objects that were found in the result of an earlier `List<object>` method (see [CreateX](#), [ListX](#), [UpdateX](#), [DeleteX](#), [SuspendX](#), [ResumeX](#) on page 70). Note that if the result of the `List<object>` method returned results in bulk format (e.g. `ListResult` or `ListEvent`), this method will always return 0 since the results cannot be accessed using the `GetResultRef` method. Instead, look at the size of the array returned for the `GetResultSet` method.

GetResultRef

ReturnPointer to search result buffer.

```
$result_ref = $obj->GetResultRef();  
  
foreach $serial_num (keys %{ $result_ref }) {  
    foreach $object_param (keys %{ $result_ref->{$serial_num} }) {  
        $param_value = $result_ref->{$serial_num}->{$object_param};  
    }  
}
```

This method provides a reference to the internal search buffer for objects that were found in result of an earlier `List<object>` method (see below). Each `List<object>` method stores results in the same internal buffer, so you should store/process the results of one search before executing a new search.

Search results are stored in double-hashed arrays, where the key for the first hash is the serial number of each object that was found, and the next hash has the parameter name as the key. One entry in the result buffer from a ListDGE method may have the format:

```
$result_ref->{<serial_number>}->{dgename} = "dge01.eng"
    ->{locationname} = "dfw"
    ->{host} = "my_server"
    ->{testcount} = 15
    ->{softlimit} = 15000
    ->{hardlimit} = 20000
    ->{serialnumber} = nnn
```

All parameter names (key for second hash) will be in lower case.

GetResultSet

Return results of a bulk search.

```
@result_set = $obj->GetResultSet();
foreach $result_item (@result_set) {
    $param_value = (split(/\|/, $result_item))[n];
}
```

This method provides a copy of the results stored in an internal search buffer for objects that were found in result of an earlier List<object> method, that returned results in a bulk format (| separated list). Each List<object> method stores results in the same internal buffer, so you should store/process the results of one search before executing a new search.

Login

Log into Traverse socket server.

```
$return_value = $obj->Login(
    [User      => $login_id],
    [Password => $login_password],
    [Timeout   => $timeout_secs]);
```

This method opens a TCP connection to *\$tcp_port* on *\$host_name_or_ip*, as defined using the *new* method. If either the *\$login_id* argument or the *\$login_pass* argument is missing, the values specified in the *new* method (if any) are used.

An optional named argument is provided to override the current timeout setting. On timeout or other connection errors, the return value is '0' and details on the error are available via the *GetErrorMsg* method. On success, a non-zero return value is provided.

This method can be used repeatedly to switch into a different user in the socket server, and assuming the new user's permissions and privileges.

Logout

Log out of the Traverse socket server.

```
$return_value = $obj->Logout;
```

This method sends a logout command to the Traverse socket server and closes the already established TCP connection to *\$host_name_or_ip*, which was defined using the *new* method.

On timeout or other connection errors, the return value is '0' and details on the error are available via the *GetErrorMsg* method. On success a non-zero return value is provided.

CreateX, ListX, UpdateX, DeleteX, SuspendX, ResumeX

```
$return_value = $obj->CreateX(  
    [Param1=>$value1,]  
    [Param2=>$value2, (...)]
```

These methods allow manipulation of different Traverse objects (X). Valid objects are:

- | | |
|--------------|--------------------|
| ■ AdminGroup | ■ Test |
| ■ UserClass | ■ Action |
| ■ Department | ■ Admin Class |
| ■ User | ■ Container |
| ■ Location | ■ DeviceDependency |
| ■ DGE | ■ Result |
| ■ Device | ■ Event |

The parameters for each object and method combination are different. Refer to [BVE FlexAPI Protocol Reference on page 17](#) for valid parameters. Not all methods are applicable to all objects. For example, a Device object can be suspended, so the SuspendDevice() method is valid, but a Location object cannot be suspended, so there is no SuspendLocation() method.

On error, for all methods, the return value is '0' and details on the error are available via GetErrMsg method. On success non-zero return value is provided. Results for ListX methods are stored in an internal array and accessed using GetResultRef method.

The following example creates a connection to localhost, logs in, creates a new DGE and logs out:

3.5 Examples

Connecting, Logging In, Creating a DGE, Logging Out

```
use Zyrion qw(Provisioning);

my $obj = new Zyrion::Provisioning(Host=>"localhost");

my $return_value = $obj->
    Login(User=>"admin", Password=>"changeme") ||
    die "ERROR: ", $obj->GetErrMsg, "\n";

$obj->CreateDGE(dgeName=>"Local DGE",
    Host=>"192.168.100.200",
```

```
locationName=>"Local Network",
softLimit=>100) ||
die "ERROR: ", $obj->GetErrorMsg, "\n";
$obj->Logout;
```

Note that the optional parameter `softLimit` was specified, but `hardLimit` was not, in which case the default value would be used.

In the following example, same login sequence is used, but now a new device is being created on the previously created DGE, and then a list of existing devices is generated:

Creating a Device for a DGE, then Listing Devices

```
use Zyrion qw(Provisioning);
my $obj = new Zyrion::Provisioning(Host=>"localhost");
my $return_value = $obj->
    Login(User=>"admin", Password=>"changeme") ||
    die "ERROR: ", $obj->GetErrorMsg, "\n";
my %param = ();
$params{deviceName} = "my test device";
$params{address} = "192.168.200.50";
$params{locationName} = "Local Network";
$params{snmpCid} = "public";
$params{comment} = "my workstation";
$params{devicetype} = "unix";
$obj->CreateDevice(%param) || die "ERROR: ", $obj->GetErrorMsg,
"\n";
$obj->ListDevice(deviceName=>'my *');
if ($obj->GetResultCount) {
    my $result_ref = $obj->GetResultRef();
    foreach my $serial_num (keys %{ $result_ref }) {
        print "device with serial number ${serial_num} ..\n";
        foreach my $object_param (keys %{
            $result_ref->{$serial_num} }) {
```

```

    $param_value =
    $result_ref->{$serial_num}->{$object_param};

    print "\t\t${object_param} = ${param_value}\n";

    }

    }

    }

$obj->Logout;

```

Note that in this case, while creating the device, instead of providing named parameters, a hash of parameters was used.

Finding Tests Without Actions Assigned

This sample script lists all devices, then checks the 'action profile' assigned to each test and prints out the ones which do not have any actions assigned.

```

$BVE = new Zyrion::Provisioning(Host=>"myhost");
$BVE->Login( user=>"joe", password=>"mypasswd");
$BVE->ListDevice(deviceName=>"*");

my %DEVICE_LIST = ();

my $RESULT_REF = $BVE->GetResultRef();

foreach my $device_serial (keys %{ $RESULT_REF }) {

    my $device_name =

    $RESULT_REF->{$device_serial}->{devicename};

    $DEVICE_LIST{$device_serial} = $device_name;

    }

## now scan through tests on each device

foreach my $device_serial (sort keys %DEVICE_LIST) {

    $BVE->ListTest(deviceName=>$DEVICE_LIST{$device_serial},
    testName=>'*');

    $RESULT_COUNT = $BVE->GetResultCount();

    next unless ($RESULT_COUNT);

    $RESULT_REF = $BVE->GetResultRef();

    foreach my $test_serial (keys %{ $RESULT_REF }) {

```

```
my $action_profile =
    $RESULT_REF->{$test_serial}->{actionname};
my $test_name = $RESULT_REF->{$test_serial}->{testname};
next unless (uc($action_profile) eq "NONE");
&info("device = $DEVICE_LIST{$device_serial} ; test =
\'$test_name\');
} # foreach test
} # foreach device
$BVE->Logout;
```

Creating a Custom SNMP Test

This is an example of creating a custom SNMP test using the API by specifying the OID directly via the API.

```
my $obj = new Zyrion::Provisioning(Host => "localhost");
my %param = ();
$param{deviceName} = "my test device";
$param{address} = "192.168.200.50";
$param{locationName} = "Local Network";
$param{snmpCid} = "public";
$param{comment} = "my workstation";
$param{devicetype} = "unix";
$obj->CreateDevice(%param);
%param = ();
$param{'deviceName'} = "my test device";
$param{'testType'} = "snmp";
$param{'subType'} = "disk";
$param{'testName'} = "Disk / Space Util";
$param{'interval'} = "300"; # seconds
$param{'units'} = "%"; # suitable unit
$param{'warningThreshold'} = "80";
```

```

$params{'criticalThreshold'} = "95";
$params{'snmpOID'} = ".1.3.6.1.2.1.25.2.3.1.6.1";
$params{'resultMultiplier'} = "1";
$params{'maxValue'} = "2048";
    # 0=rate, 1=percent, 2=delta, 3=rate
    # 4=deltapct, 5=ratepct
$params{'resultProcessDirective'} = "1";
$obj->CreateTest(%params);
    
```


4.1 Overview

The plugin monitor functionality in Traverse allows creating new monitors in Java or any other programming language such as C, perl, shell, etc. The system treats such plugin monitors as an integrated component of Traverse and provides a similar multi-threaded framework as it uses internally for its own monitors.

Each plugin monitor has an associated XML configuration file describing the test type, default thresholds and various display parameters. The configuration files are installed in the `$TRAVERSE_HOME/plugin/monitors/` directory and the actual plugin monitor file is installed in a subdirectory under the `plugin/monitors` directory. The name of the subdirectory must match the monitor type specified in the configuration file.

4.2 Adding A New Test Type

Each test configured in Traverse is assigned a type and sub-type. The test type and sub-type combination serves as the key for global default information that is read from various configuration files. If Traverse is unable to locate the configuration information for a particular test type and sub-type, it will be ignored (with an error message logged). Such configuration information is loaded from `$TRAVERSE_HOME/etc/TestTypes.xml` and other plugin configuration files (described in the sections that follow).

When creating new (plugin) monitors, you will need to create a unique test type and sub-type for that monitor and provide various default values and other parameters. The entries in `$TRAVERSE_HOME/etc/TestTypes.xml` or other directories should not be edited (unless you are instructed to edit them by Zyrion Customer Support) as it may adversely affect or cause failure of Traverse components. Any changes made to these directories may also be lost when a new version of Traverse is installed. All user customizations are expected to be placed in `$TRAVERSE_HOME/plugin` and its subdirectories.

Sample TestTypes.xml Entry

```
<testtype>

  <displayName>Current Temperature</displayName>
  <displayCategory>application</displayCategory>
  <subType>temperature</subType>
  <units>degrees C</units>
  <severityAscendsWithValue>true</severityAscendsWithValue>
  <defaultWarningThreshold>100</defaultWarningThreshold>
  <defaultCriticalThreshold>120</defaultCriticalThreshold>
  <shadowWarningThreshold>100</shadowWarningThreshold>
  <shadowCriticalThreshold>120</shadowCriticalThreshold>
  <slaThreshold>120</slaThreshold>
  <testInterval>180</testInterval>
  <showAsGroup>true</showAsGroup>

  <testField>
    <fieldName>city</fieldName>
    <fieldDisplayName>City</fieldDisplayName>
    <isRequired>true</isRequired>
    <isPassword>false</isPassword>
    <defaultValue>Muskogee</defaultValue>
  </testField>

  <testField>
    <fieldName>state</fieldName>
    <fieldDisplayName>State/Province</fieldDisplayName>
    <isRequired>true</isRequired>
    <isPassword>false</isPassword>
    <defaultValue>OK</defaultValue>
  </testField>

  <testField>
    <fieldName>country</fieldName>
    <fieldDisplayName>Country</fieldDisplayName>
    <isRequired>true</isRequired>
    <isPassword>false</isPassword>
    <defaultValue>US</defaultValue>
  </testField>

</testtype>
```

The testtype element includes the following child elements:

Table 4-1. XML Testtype Child Elements

Child Element	Description
<code>displayName</code>	A user-friendly name that is used when creating a report or referring to a specific <code>testtype</code>
<code>displayCategory</code>	This setting defines the column that the test result should be in on the summary pages in the Web application. Valid values are <code>network</code> , <code>system</code> , and <code>application</code> .
<code>subType</code>	This is a string that uniquely identifies the <code>testtype</code> to the Traverse software. You can choose whatever string you want, with some restrictions. The subtype must be unique to the monitor that the test is running on, and can only contain alphanumeric characters.
<code>units</code>	The units for the test measurement. This will be used in reports and event and summary displays. If the particular test does not have a suitable unit, use a space as the unit.
<code>severityAscendsWithValue</code>	This is used to indicate a severity direction for test values, and has the following possible values: <code>true</code> <code>false</code> or <code>static</code> If the value is <code>true</code> , then the status being tested becomes more critical as the test value rises. When the value is <code>static</code> , you can set discrete threshold values for warning and critical.
<code>defaultWarningThreshold</code>	This is the default end user warning threshold for this test type. If the <code>severityAscendsWithValue</code> is 'static', then you can specify a comma separated set of numbers using the following syntax: 1,3,5,8-20
<code>defaultCriticalThreshold</code>	This is the default end user critical threshold for this test type.
<code>showAsGroup</code>	Group tests of the same sub-type together in the Web application during autoDiscovery of SNMP tests for a device.
<code>shadowWarningThreshold</code>	The default admin warning threshold for this test type. Typically this value will be same as <code>defaultWarningThreshold</code> .

Table 4-1. XML Testtype Child Elements

Child Element	Description
<code>shadowCriticalThreshold</code>	The default admin critical threshold for this test type. Typically this value will be same as <code>defaultCriticalThreshold</code> .
<code>slaThreshold</code>	The default SLA threshold for this test type.
<code>testInterval</code>	The default interval, in seconds, for running this test.
<code>testField</code>	<p>This element defines a specific attribute for the test. A testtype can have 0 or more test fields. Each testfield should have the following child elements:</p> <p><code>fieldName</code> - This will be used as key for the field value when it's passed to the test.</p> <p><code>fieldDisplayName</code> - A user friendly name for the field that will be used by the Web application when creating or updating tests.</p> <p><code>isRequired</code> - This element indicates whether or not the a value is required to be given for the field when creating or updating the test.</p> <p><code>isPassword</code> - This indicates whether or not the field is a password field. The Web application will ask for verification of password fields when creating or updating a test.</p> <p><code>defaultValue</code> - A default value that will be presented to a user when creating the test.</p>

4.3 Creating A New Plugin Java Monitor

Traverse allows you to extend its functionality by writing plugin monitors in Java. Such monitors can collect information from various applications and/or devices. This involves creating the monitor, packaging it and creating a corresponding configuration file.

Configuration File Format

Traverse uses an XML file called a "test descriptor" to describe settings for plugin tests. Here is an example test descriptor that might be used to describe a plugin that monitors weather information.

Weather Information Plugin Test Descriptor

```

<monitor type="weather" pluginType="java"
resource="com.weatherwatchers.netvigilplugin.WeatherPlugin">

  <testtype>

    <displayName>Current Temperature</displayName>
    <displayCategory>application</displayCategory>
    <subType>temperature</subType>
    <units>degrees C</units>
    <severityAscendsWithValue>true</severityAscendsWithValue>
    <defaultWarningThreshold>100</defaultWarningThreshold>
    <defaultCriticalThreshold>120</defaultCriticalThreshold>
    <shadowWarningThreshold>100</shadowWarningThreshold>
    <shadowCriticalThreshold>120</shadowCriticalThreshold>
    <slaThreshold>120</slaThreshold>
    <testInterval>180</testInterval>

    <testField>
    <fieldName>city</fieldName>
    <fieldDisplayName>City</fieldDisplayName>
    <isRequired>true</isRequired>
    <isPassword>false</isPassword>
    <defaultValue>Muskogee</defaultValue>
    </testField>

    <testField>
    <fieldName>state</fieldName>
    <fieldDisplayName>State/Province</fieldDisplayName>
    <isRequired>true</isRequired>
    <isPassword>false</isPassword>
    <defaultValue>OK</defaultValue>
    </testField>

    <testField>
    <fieldName>country</fieldName>
    <fieldDisplayName>Country</fieldDisplayName>
    <isRequired>true</isRequired>
    <isPassword>false</isPassword>
    <defaultValue>US</defaultValue>
    </testField>

  </testtype>

</monitor>

```

The first element is the `monitor` element. The `monitor` element defines what monitor the different tests belong to. There are three attributes for the `monitor` element:

Table 4-2. Monitor Element Attributes

Attributes	Description
<code>type</code>	This defines a type name for the plugin, and the type of monitoring it does. The value of <code>type</code> will show up in the DGE status line when displaying the testing queues and monitor status, and will be used in the Web application
<code>plugintype</code>	This attribute describes the type of plugin. For a Java plugin monitor, this parameter should be set to <code>java</code> .
<code>resource</code>	This is the name of the resource of that should be used to do the tests. For a <code>plugintype</code> of <code>java</code> , this should be the fully qualified name of a Java class file that implements the <code>NetvigilPlugin</code> interface.

The configuration file also requires a `testType` definition as described above. You should make a different XML test descriptor for each type of monitor you want to create. To group multiple tests that belong to the same general test type, each monitor `type` can have multiple `testType` definitions with `subType` defined for each test. These can be contained within a single monitor descriptor or spread across separate XML files.

Writing The Plugin Class

Your plugin class should be able to be run under Sun JRE 1.5. Your plugin class must implement the `NetvigilSimplePlugin` interface, or the `NetvigilBatchPlugin` interface. See the javadoc for more information. The `NetvigilSimpleInterface` should be used when only small amounts of plugin tests will be provisioned, or for tests with long testing intervals. The `NetvigilBatchPlugin` should be used when large sets of tests can be run at one time, and where the tests require some expensive operation before they run, such as opening a connection.

Plugin tests can then be created in the Web application or Traverse socket interface. Once created, the plugin tests are stored in the provisioning database, with each of the `testField` values for that test, with the `fieldName` as a key. The DGE loads the tests from the database, and after it has determined that the testing interval has passed, adds the test to a test queue, indicating that the test should be run.

If the plugin implements the `NetvigilSimplePlugin` interface, the DGE will create a new instance of your `NetvigilSimplePlugin` subclass and call the `doTest` method for each plugin test in its test queue, passing a `java.util.Properties` object with the `testField` values. The value returned by `doTest` will be stored in the DGE database, and will be used for reports and status. If the value returned is `RESULT_UNKNOWN` or `RESULT_FAILED`, the DGE will call `getErrorMessage` and will put any returned error message in the Traverse error log, so the Web application user can determine the reason for a failed test.

If the plugin implements the `NetvigilBatchPlugin` interface, the DGE will create one instance of the plugin when it starts up, and will call `addTest` on that instance for each plugin test in its test queue, again passing a `java.util.Properties` object with the `testField` values. The DGE will call `addTest` until the test queue has no more tests of the plugin type, or until the number of tests specified by `getMaxBatchSize` has been reached. After this, the DGE will call the `runBatch` method of the plugin object. When `runBatch` returns, the DGE will call `getTestResults` to get the results of the batch test. The order of results returned in the array by `getTestResults` must match the order that the DGE called `addTest`. The DGE will take the results and store them in the DGE database, where they can be used in reports and status displays. If any of the results has a value of `RESULT_UNKNOWN` or `RESULT_FAILED`, the DGE will call `getErrorMessage` with the index of the result in the array returned by `getTestResults`. If the error message returned is not empty, it will be logged in the Traverse error log.

Configuring the Plugin Package

Once you're done creating your class, create a `.jar` file for it and any other required classes. Create an XML test descriptor as described above for your class. Place the test descriptor in `$TRAVERSE_HOME/plugin/monitors`. Make a directory in a `$TRAVERSE_HOME/plugin/monitors` called `<type>/lib`, where `<type>` is the type of monitor in your XML test descriptor. So, for the weather example described in [Weather Information Plugin Test Descriptor on page 81](#), you should create a directory called `$TRAVERSE_HOME/plugin/monitors/weather/lib`. Place the `.jar` file you just created in the `lib` directory. If Traverse is installed in a distributed environment (multiple hosts), the plugin package and test descriptor file should be installed on each host running Traverse.

Provisioning Plugin Tests

The Web application and DGE components will need to be restarted before the new monitor is usable. When Traverse starts, it will scan the monitor plugin directory for XML and `.jar` files. It will add the tests described by the XML file to its list of test descriptions, and will add the `.jar` files found in `<type>/lib` to the Java CLASSPATH. If an XML file has an error in it, a message will be written to the error log, and the XML file will be ignored. Each different XML file results in a separate test queue in the Traverse DGE.

To create a plugin test in the Web application, choose Create New Custom Tests from the Manage Tests page. The test settings you defined in the plugin XML file should show up on this page. Fill in the form fields, select the Provision? checkbox next to each test you want to provision, and click the Provision Tests button at the bottom of the page. You should be able to update or delete your newly created plugin tests in the same manner you update and delete the other test. You can also create and update tests through the Traverse socket server. Just type `help Test.create` or `help Test.update` on the Traverse socket server command line to see the specifics for creating or updating a plugin test.

4.4 Creating A New Plugin Script Monitor

The following section describes how to create a new plugin script monitor.

Configuration File Format

Traverse uses an XML file to describe settings for script plugin monitors. Here is an example test descriptor that might be used to describe a plugin script that monitors weather information:

```
<monitor type="weather" pluginType="script">
  <!--
  Insert a testType element here.
  -->
  <script type="weather" subType="temperature">
    <rootScript>gettemp.pl</rootScript>
    <timeout>10</timeout>
    <parameters>--country=${country} --state=${state}
```

```

    --city=${city} </parameters>

    </script>

</monitor>

```

The first element is the `monitor` element. The `monitor` element defines what monitor the different tests belong to. There are two attributes for the `monitor` element:

Table 4-3. Monitor Element Attributes

Attributes	Description
<code>type</code>	This defines a type name for the plugin, and the type of monitoring it does. The value of <code>type</code> will show up in the DGE status line when displaying the testing queues and monitor status, and will be used in the Web application
<code>plugintype</code>	This attribute describes the type of plugin. For a script plugin monitor, this parameter should be set to <code>script</code> .

The `monitor` element also requires a `testType` definition as described above.

The second element is the `script` element. This element describes the way the script should be run. The `script` element has two attributes, `type` and `subType`, that will associate the script with a test type. The `type` attribute for the script should have the same value as the `type` attribute of the `monitor` element. In this case, they're both `weather`. The value of the `subType` attribute should match the `subType` attribute of one of the `testType` elements owned by the monitor. Our script will get the temperature, so we want it to be associated with the temperature test type, so we give its `subType` the same value, `temperature`, as the `subType` for the temperature test type.

The next two child elements are fairly straightforward. The `rootScript` element gives the name of the script to run, and the `timeout` element gives the maximum number of seconds to wait for the script. You should give a timeout of less than 60 seconds for your script, so that the Traverse monitor running your script can return in a timely manner if your script hangs for some reason. Timeout values of zero or less will be interpreted as a 60-second timeout.

The final child element is the `parameters` element, which defines how arguments are passed to the script. You can enter any text for the `parameters` object, and you can also use `testField` placeholders to indicate where `testField` values should be passed. To use a placeholder, simply enter `${`, followed by the `fieldName` of a `testField` for the `testtype` your script plugin is handling, and end the placeholder with a `}`.

In addition, the following variables can also be used in the `parameters` element of the configuration file:

- `${device_name}`
- `${device_address}`
- `${device_vendor}`
- `${device_snmp_cid}`
- `${device_location}`
- `${test_serial_number}`
- `${test_user_critical_threshold}`
- `${test_shadow_critical_threshold}`
- `${test_type}`
- `${test_units}`
- `${device_serial_number}`
- `${device_model}`
- `${device_type}`
- `${device_snmp_version}`
- `${test_name}`
- `${test_user_warning_threshold}`
- `${test_shadow_warning_threshold}`
- `${test_sla_threshold}`

When Traverse calls your script, it will replace the placeholders with the values given when a test was provisioned. Based on the example above, if you provisioned a test for London, England, Traverse would call the script with the following arguments:

```
--country=England --state= --city=London
```

If you don't provide a `parameters` element, the script is called without any arguments.

Writing The Plugin Script

You can write your script in any language you want. When called with the set of arguments you defined in the parameters element of your plugin XML file, your script should run a test based on the arguments. If the test was completed successfully, your script should print a zero or a positive integer on standard out that corresponds to the value determined by testing. If your test failed for some reason, you can print one of the following error codes: -1, to indicate that the test failed for an unknown reason, or -2 to indicate that the test failed for a known reason.

You can also pass out debugging or error information from your script. Any lines beginning with the string "DEBUG:" will be logged to the Traverse debug log (this log is turned off by default in the typical Traverse installation. Contact Traverse support for instructions to turn it on.). Any lines beginning with the string "ERROR:" will be logged to the Traverse error log.

Once you're done writing your script, you should test it out separately on the command line to be sure it works. Next, place your script in `$TRAVERSE_HOME/plugin/monitors/<test_type>` directory (where `<test_type>` is the type name specified in the configuration file. Place the XML test descriptor in `$TRAVERSE_HOME/plugin/monitors`. If Traverse is installed in a distributed environment (multiple hosts), the monitor script and configuration file should be installed on each host running Traverse. The Web application and DGE components will need to be restarted before the new monitor is usable.

Sample Plugin Monitor with Discrete Thresholds

This is an example of a sample atmosphere pressure monitor which uses discrete thresholds.

Creating a Plugin Monitor with Discrete Thresholds

- 1 Create a test type definition file: `plugin/monitors/my_atmosphere.xml` with the following contents:

```

<monitor type="atmosphere" pluginType="script">
<testtype>
<displayName>Atmospheric Pressure</displayName>
<displayCategory>application</displayCategory>
<subType>pressure</subType>
<units>psi</units>
<severity_ascends_with_value>discrete</severity_ascends_w
ith_value>
<defaultWarningThreshold>2,5</defaultWarningThreshold>
<defaultCriticalThreshold>4,8-10,99</defaultCriticalThresho
ld>
<shadowWarningThreshold>2,5</shadowWarningThreshold>
<shadowCriticalThreshold>4,8-10,99</shadowCriticalThresho
ld>
<slaThreshold>8-10</slaThreshold>
<testInterval>60</testInterval>
</testtype>

<script type="atmosphere" subType="pressure">
<rootScript>run.sh</rootScript>
<parameters></parameters>
<waitForTerminate>true</waitForTerminate>
<timeout>15</timeout>
</script>
</monitor>

```

Note how the thresholds have been specified as discrete values. If the polled result is 2 or 5, the test will be in warning state, critical is 4,8,9,10 and 99. Everything else is OK.

- 2 Now create the monitor in the `plugins/monitors/` directory under a directory with the same name as the test type, and name it as indicated in the test type definition above (`plugins/monitors/atmosphere/run.sh`)

```

#!/bin/sh

#if [ -f "/tmp/atmosphere.dat" ]; then

cat /tmp/atmosphere.dat

else

echo 0

fi

```

- 3 Now restart the Web application, and then provision the test:
- 4 Log in as end user.
- 5 Click on Administration > Devices > **Tests**.
- 6 Click on **Add New Standard Tests**.

- 7 Check **Atmosphere**.
- 8 Enable the test, and make sure that “discrete” is selected as a severity option.
- 9 Submit the form.

To test this, enter values into `/tmp/atmosphere.dat` and you can see the test status change in each polling cycle:

```
echo 99 >/tmp/atmosphere.dat
echo 5 > /tmp/atmosphere.dat
echo 6 > /tmp/atmosphere.dat
```

4.5 Extending the Message Handler

Users can extend the Message Handler to handle additional message sources and write custom rulesets by creating additional configuration files and storing them in the plugins directory under `$TRAVERSE_HOME/plugin/messages/`. Additional data sources should be defined in configuration files named as `nn_src_yyy.xml` while additional rulesets should be named `nn_rule_yyy.xml` (`nn` is a number and `yyy` is any freeform text).

As an example, you can add new log files to be monitored and a trap handler listening on port 2162 by creating the following two files in the `$TRAVERSE_HOME/plugin/messages/` directory:

00_src_logs.xml

```
<source type="file" name="mylog">
<enabled>>false</enabled>
<input>/var/log/mylogs</input>
</source>
<source type="file" name="apacheErrLog">
<enabled>>true</enabled>
<input>/apache/logs/httpd.error</input>
</source>
```

00_src_traps.xml

```
<source type="trap" name="traps2">
    <enabled>true</enabled>
    <port>2162</port>
    <performHostnameLookup>false</performHostnameLookup>
</source>
```

The format for the rule files is described in the *Traverse User Guide*.

Remember to restart the Message Handler after editing or creating new files.

5.1 Overview

You can override the standard Traverse authentication methods by creating your own plugin Java classes or scripts. This allows the use of custom authentication databases or other site-specific authentication methods to control access to Traverse. Note that although it is possible to use an external authentication source, the authorization information (permissions, limits, etc.) are still stored in Traverse's provisioning database. So it will be necessary to create the login ID on Traverse database even though that login is authenticated from an external database.

Additionally, Traverse provides the facility for integrating with a Web portal and using the Web portal's authentication mechanism.

NOTE: If you change the authentication mechanism, the changes are applied only to users created after the change was made. To use the new authentication method for users created before the authentication mechanism changed, you must change their passwords using either the Web application or the BVE TCP server (see [userClass.update on page 50](#)). Otherwise, older users will continue to be authenticated by the old mechanism (e.g., the Traverse internal password database).

Authentication Plugin Java Class

If you want to create a java class for authentication, your plugin class must implement the `NetvigilPluginAuthentication` interface. See the javadoc for `NetvigilPluginAuthentication` for more information.

Your implementation of the Java class `NetvigilPluginAuthentication`.`getAuthenticationString` should take the user login name and password, and create an authentication string, such as an encrypted password, from this information. The `Properties` argument to `getAuthenticationString` is reserved for future use. When a user logs in, your implementation of `NetvigilPluginAuthentication`.`authenticate` will be given the user login name, the authentication string for that user that was created by `getAuthenticationString`, and the password the user

gave when he or she tried to log in. As with `getAuthenticationString`, the `Properties` parameter to `authenticate` is reserved for future use. Your version of `authenticate` should use this information to determine whether or not the user should be allowed to log in. If the user should be allowed to login, your version of `authenticate` should return `true`.

As an example, for a Java class which does authentication using `rot13` to encrypt a given password, you should choose a unique string to identify the authentication method for your class. This string will be stored in the database to indicate the type of authentication method to use with a given authentication string. The strings `clear`, `des`, and `script` are reserved, but you can use any other unique string to identify your plugin authentication method.

Finally, you must use the unique string along with the name of your plugin class in the `authentication` section of `$TRAVERSE_HOME/etc/emerald.xml` to identify your plugin class as the class to use for authentication. For example, if you wanted to use the `rot13` class mentioned above for authentication, you could choose the string "rot13" as the identifier, and modify the `authentication` section in `emerald.xml` so that it looks like this:

```
<authentication
    method="rot13"
    class="Rot13Authentication"
    execute=""
    parameters=""
/>
```

Leave the `execute` and `parameters` attributes in the `authentication` section empty. They're reserved for plugin authentication using scripts (described in [Authentication Plugin Script on page 93](#)).

Once you're done writing your class, create a `.jar` file for it and any other required classes and place them in `$TRAVERSE_DIR/plugin/auth` directory. Your plugin class, and any third party `.jars` you've included, must work under Sun JRE 1.5.

Authentication Plugin Script

You can also specify a script, program or batch file to use for authentication. Traverse will run this program and pass it the user's login name and password as arguments. Following the convention of using a zero return code for successful program execution, your script must return a zero value to indicate that authentication was successful. You can specify the format that the arguments are passed to your program.

Here's an example Perl script (**auth.pl**) that will only let a user named "jane" log in, and only if she gives the password "secret".

Sample Login Authentication Script

```
#!/usr/bin/perl

if($#ARGV != 1) {
    print STDERR "not enough arguments!\n";
    # exit with a non-zero
    exit 2;
}

# get the username and password from the arguments
#
# we've set up our parameter string so that username
# is the first argument, and password is the second
#
$username = $ARGV[0];
$password = $ARGV[1];

if($username eq "jane" && $password eq "secret") {
    # return 0 so that jane can log in to Traverse
    exit 0;
} else {
    # return a non-zero failure code, since the username
    # and/or password was wrong.
    exit 1;
}
```

```
}
```

Once you're done with your script, place it in the Traverse plugin authentication directory (`$TRVERSE_HOME/plugin/auth`). To instruct Traverse to use your script for authentication, you'll need to modify `$TRVERSE_HOME/etc/emerald.xml`. Update the authentication element, which initially may look like this:

```
<authentication  
  
    method="des"  
    class=""  
    execute=""  
    parameters=""  
  
</>
```

Change this so that the `method` attribute is `script`. This will tell Traverse that you want to do authentication with a script. Leave the `class` attribute empty, since that's only used for plugin authentication using a Java class (described in [Authentication Plugin Java Class on page 91](#)). Place the name of your script in the `execute` attribute. Use the `parameters` attribute to specify the order that the username and password should be passed to your script, along with any other flags you want passed. You can use the special variables `${username}` and `${password}` as placeholders for the username and password respectively. For example, you may want your script to take GNU-style long parameters, so you could set the `parameters` attribute to something like this:

```
--username=${username} --password=${password}
```

Since our example script doesn't use any flags for the username and password, we'll use `${username} ${password}` for the parameters. The authentication section of **emerald.xml** would look like this after we're done:

```
<authentication  
  
    method="script"  
    class="class=""  
    execute="auth.pl"  
    parameters="${username} ${password}"  
  
</>
```

Note that any existing Traverse users will still continue to be authenticated using the older authentication method (since the authentication method is stored with each user entry in the database). To switch them to the new scheme, simply change their password once. This allows you to keep the password for "superuser" tied to the local authentication scheme and not dependent on an external resource or database.

WARNING: Any parameters passed to the plugin script you specify may be viewed by anyone on your system with the ps command during the time it takes the script to execute.

5.2 Web Portal Authentication

You can bypass the initial login page in Traverse by directly encoding the username and password information in the URL and encrypting this information using a shared key. This mechanism allows a user to access Traverse via some other portal where he/she has already been authenticated.

- 1 Edit `$TRAVERSE_HOME/webapp/WEB-INF/web.xml` and change the shared key in `<param-name>externalLoginKey</param-name>`
- 2 Copy `$TRAVERSE_HOME/utills/externalWebLogin.cgi` to your Web portal.
- 3 Edit this script and set the shared key, as well as the mechanism to get the department, username, password (can be changed to extract from the HTTP environment depending on your setup).
- 4 Set `maxPages` to 1 to limit the user to only view the one page that the URL connects to, else leave as -1 for full access.

This allows displaying just one page (e.g making one report publicly available) and not allowing a full login.

5.3 Architectural Description

If you've specified a plugin java class to use for authentication, when a user password is created or changed, your implementation of the `NetvigilPluginAuthentication.getAuthenticationString` method will be called. The authentication string this method returns will be stored in the provisioning database, along with the unique string you picked to identify the authentication method. Traverse will use the unique string as a key to find and load your plugin

authentication class. When the user tries to login to a Traverse application, this authentication string will be retrieved from the database, and it, along with the password the user gave and the user login name will be passed to your `NetVigilPluginAuthentication.authenticate` method. If your `authenticate` method returns true, the user will be allowed to login. Conversely, if `authenticate` returns false, the user won't be able to login.

If you've told Traverse to use a plugin script, when the user logs in, Traverse will take the user login name, password and the `parameters` attribute from **emerald.xml**, and will replace the placeholders in the `parameters` attribute with the login name and password. It will then look in the authentication scripts directory for the script named in the `execute` attribute in **emerald.xml**, and will execute the script with the updated `parameters` attribute. If the script runs successfully, and returns a zero exit code, Traverse will allow the user to log in. If Traverse can't run the script, or the script returns with a non-zero exit code, the user will not be allowed to login.

6.1 Overview

Traverse provides a strong plugin framework which enables you to extend the native capabilities. While Traverse provides various common notification mechanisms, such as email, pager, trouble ticket interface and SNMP traps, the plugin action framework allows custom notification and action development as needed. These custom plugin actions seamlessly integrate with the Traverse Action Policy module which allows notification policies controlled by time of day, number of polls before activation, etc.

6.2 Creating New Plugin Actions

Creating plugin actions requires two components:

- XML configuration/definition
- The script itself

Before a plugin action is available to users, you need to create a configuration file defining the location of the script to call, and what parameters need to be passed to this script. The configuration file needs to be created under `$TRAVERSE_HOME/plugin/actions` directory. There are no restrictions on what the configuration file can be named. However, the file must have a `.xml` extension, as only `.xml` files are scanned for configuration information.

Here is a sample configuration file (**writeToFile.xml**):

writeToFile.xml

```
<?xml version="1.0" standalone="yes"?>
<!--
    All plugin "script" action configuration should be enclosed
    in an <ActionScriptConfig>.. </ActionScriptConfig> block
-->
```

```

<ActionScriptConfig>
  <!--
    This is the name of the script action that will appear in
    the drop-down list within the action profile management page on
    the web application. This name should be unique. It should not
    match the name of any other existing native or custom action.
  -->

  <name>My Custom Script</name>

  <!--
    This is the script/batch file/application to be executed.
    Use only the name of the script/application, and do not include
    the path. Traverse looks for this script under the
    $TRAVERSE_HOME/plugin/actions directory.
  -->

  <rootScript>writeToFile.sh</rootScript>

  <!--
    The parameters to pass to the script when executing it. See
    below for a list of variables that you can use as parameters.
    Parameters can be specified in multiple lines. At execution
    time, they are concatenated into a single line.
  -->

  <parameters>
  -d ${device_name}
  -t ${test_name}
  -s ${current_user_severity}
</parameters>

  <!--
    When executed, should Traverse wait for the script it to
    terminate? Possible values:true or false.
  -->

  <waitForTerminate>true</waitForTerminate>

  <!--
    If waitForTerminate is true, how long (in seconds) should
    Traverse wait before aborting the script? If set to 0 or a
    negative value, the application will wait indefinitely for the
    script to terminate.
  -->

  <timeout>10</timeout>

  <!--
    If true, the output from the script will be added to the
    device comment on the web application. Enabling this option
    automatically sets waitForTerminate to true.
  -->

```

```
<addOutputToComment>>false</addOutputToComment>
</ActionScriptConfig>
```

The following variables can be used in the parameters section of the configuration file:

- `${department_name}`
- `${recipient}`
- `${device_serial_number}`
- `${device_model}`
- `${device_type}`
- `${device_snmp_version}`
- `${current_user_severity}`
- `${current_sla_severity}`
- `${action_item}`
- `${action_class}`
- `${timestamp}`
- `${device_name}`
- `${device_address}`
- `${device_vendor}`
- `${device_snmp_cid}`
- `${device_location}`
- `${current_shadow_severity}`
- `${time_in_state}`
- `${action_profile}`

The following additional variables are available for events triggered by polled test results only:

- `${test_name}`
- `${test_user_warning_threshold}`
- `${test_shadow_warning_threshold}`
- `${test_sla_threshold}`
- `${test_sub_type}`
- `${result_value}`
- `${test_serial_number}`
- `${test_user_critical_threshold}`
- `${test_shadow_critical_threshold}`
- `${test_type}`
- `${test_units}`
- `${event_reason}`

The following variables are specific to events triggered by the Message Handler (for syslogs, log files, traps, etc.):

- `${message}`
- `${message_source}`
- `${ruleset_description}`

- `${original_message}`
- `${message_type}`

As a security precaution, the actual script must be in the `plugin/actions` directory. If the command line tool is in a different directory, you can either create a wrapper script/batch file that calls the real program, or create a symbolic link (UNIX only) to the real program into `plugin/actions`.

Here is a sample script that corresponds to the sample configuration file provided above (`writeToFile.sh`):

writeToFile.sh

```
#!/bin/sh

#

time=`date '+%Y%m%d-%H:%M'`

echo "time:$time, device: $2, test: $4, severity: $6" \

>> /tmp/severity.log
```

The configuration file, and the script, need to be installed under `plugin/actions` directory on all hosts that are running Traverse application. Before the new action is available, the Web application and DGE components will need to be restarted. Once the configuration file has been loaded, it will show up on the drop-down list in action profile management page within the Web application. To use the newly added plugin action, you first need to create an action profile that uses this script.

Creating an Action Profile

- 1 Create an action profile via Administration > Actions > **Create An Action Profile** (or update an existing profile).
- 2 From the **Notify Using** drop-down list, you should be able to select the script. The name displayed on the list will correspond the `<name>...</name>` parameter in the configuration file (`My Custom Script`).
- 3 The **Message Recipient** field can be left empty and the rest of the parameters set as you see fit.
- 4 Apply this action profile to various tests as required.

For example, if an action profile containing this sample action is assigned to a test called `My Test` for a device called `My Device`, when the action profile is triggered for `warning` severity, the DGE component executes this script as:

```
/$TRaverse_HOME/plugin/actions/writeToFile.sh -d "My Device" -t
"My Test" -s "warning"
```

and waits 10 seconds for the process to complete. Upon successful execution `/tmp/severity.log` should have an entry that looks like this:

```
time:2002nnnn-hh:mm, device: My Device, test: My Test, severity:
warning
```

You can use the same script for multiple actions (for example, with different parameters). In order to accomplish this, you will need to create multiple plugin action configurations that correspond to the same script.

6.3 Examples

Reboot Router

For example, if the CPU utilization on a router stays consistently at 80%, the following plugin can be used to reboot a router. The files would need to be placed in the `$TRaverse_HOME/plugins/actions` directory.

`plugin/actions/rebootRouter.xml`

```
<?xml version="1.0" standalone="yes"?>
<ActionScriptConfig>
  <name>Reboot Router (via telnet)</name>
  <rootScript>rebootRouter.pl</rootScript>
  <parameters>${device_address}</parameters>
  <addOutputToComment>>false</addOutputToComment>
  <waitForTerminate>>true</waitForTerminate>
  <timeout>60</timeout> <!-- seconds -->
</ActionScriptConfig>
```

plugin/actions/rebootRouter.pl

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

# DESCRIPTION:

# log into a cisco router, switch to enable mode

# and reboot it

use Net::Telnet;

my $device_address = $ARGV[0];      # Passed from Traverse

my $login_user     = "username";    # SET THIS

my $login_pass     = "password";    # SET THIS

my $enable_pass    = "enable";      # SET THIS

my $socket = new Net::Telnet (%PARAM);

$socket->open(Host => $device_address, Port => 23);

$socket->login($login_user, $login_pass);

$socket->print("enable");

$socket->print($enable_pass);

$socket->print("reload in 2 automated Traverse action");

$socket->print("exit");

$socket->close;
```

NOTE: This script sample does not include error management. It only highlights the basic commands for logging into and rebooting a Cisco router.

6.4 Extending the Action Framework

The action framework can be extended easily using the Plug-in Framework to run any external program. The device name and test information can be passed to the external program to build very flexible actions (which can then use the API to query the state of another device and test before executing a corrective action).

RT Trouble Ticketing Plugin

This integration package adds a new custom action to the drop-down list of actions available to a user of the Traverse Web application. The action can be configured to trigger after certain number of test cycles, repeat after several test cycles, and trigger during certain hours of the day, like any Traverse action. Once triggered, the script connects to an existing RT (version 2.x) system and searches for a ticket in the specified queue matching certain subject (created using device and test name). If found, the ticket is updated with new information. Otherwise a new ticket is created and the URL to the ticket is added to the device comment.

NOTE: This plugin needs to be licensed separately. Contact Zyrion Customer Support for more information.

Prerequisites

Before installing this package, the following tasks need to be completed:

- If you have multiple locations defined in your Traverse environment, decide which locations should have the ability to open tickets in RT. As each location may have multiple DGE, this will assist you in compiling a list of DGE where the package will need to be installed.
- This package uses RT Perl API and requires the RT Perl modules to be functional. In order for this tool to function properly, RT must be installed and configured properly on each DGE (if the Web application is running on a separate host, there is no need to install RT on that host). Install RT under its default location `/opt/rt2`, or install it at a location of your choice, and create a symbolic link from `/opt/rt2` to that directory. Instructions for installing RT are available from www.bestpractical.com/rt.
- Copy `etc/config.pm` from your RT host to `/opt/rt2/etc/config.pm` on all the DGE. Edit `/opt/rt2/etc/config.pm` and update `$DatabaseHost` to point to your RT host. You also may want to update `$LogDir`, or create the directory specified and make sure that the directory permissions are set up properly.
- Create a new login for Traverse into RT database (via WebRT). Set an appropriate username (for example, `traverse@your.domain`) but make sure to leave the email field blank. This will make sure that when a new ticket is created, no auto-replies will be sent (if there is such a script configured for the queue you will be using).

- Make sure the newly created user has permissions to create new tickets and add comments to existing tickets.
- You may have to configure your RT database for remote access. By default, when using MySQL for RT database, the database user (specified in `/opt/rt2/etc/config.pm`, variable `$DatabaseUser` is only allowed access from localhost. Before this custom action can create/update tickets, it will need to be allowed access. For MySQL, this involves connecting to the `rt2` database (or the database name specified in `/opt/rt2/etc/config.pm`) locally on the RT host as root, and using:

```
GRANT SELECT, INSERT, CREATE, INDEX, UPDATE, DELETE ON rt2.* TO
rt_user@n.n.n.n;
```

- where `n.n.n.n` is the IP address of each DGE. When using Postgres database, you will have to make necessary additions to `data/pg_hba.conf` file. Please refer to configuration documents for the respective database vendors (MySQL or PostgreSQL) for additional details.
- Make sure the RT Perl modules are working properly. The `test-rt.pl` test script should be able to search and display all new/open tickets in your RT system (replace `YOUR_QUEUE_NAME` in the script with a valid queue name). Create and run the script from each DGE to verify proper installation and communication with RT.

Installation

Installing RT Trouble Ticketing Plugin

- 1 Copy the installation package to each DGE (that should have the custom action, and also to the host running the Web application. Store it in a temporary location:
- 2 Extract the files and start installation:

Windows:

Double-click the installation executable.

UNIX:

```
cd /tmp
gunzip -c integ-rt2-n.n.tar.gz | tar xvf -
cd integ-rt2-n.n
perl ./install.pl
```

- 3 Provide answers to the requested questions. The installation process will copy the integration package into appropriate location under Traverse installation directory.
- 4 You will need to restart the DGE process and Web application at a convenient time before the action will be visible in the drop down list (in Web application), or can be executed by a DGE.

Configuration on Windows

- 1 To use the newly added plugin action, you first need to create an action profile that uses this script. Create an action profile via Administration > Actions > **Create New Action** (or update an existing profile). From the Notify Using drop-down list, you should be able to select the script. The name displayed on the list will correspond to the `<name> . . . </name>` parameter in `$TRAVERSE_HOME/plugin/actions/createTicketInRT.xml` file. The **Message Recipient** field can be left empty and rest of the parameters set as you see fit. Now apply this action profile to various tests as required.

- 2 If you wish to create tickets in different queues, you will need to create two different plugin actions - one each for the two RT queue:

- a Stop Traverse:

Start > Programs > Zyrion's Traverse > **Stop Traverse**

- b In `$TRAVERSE_HOME/plugin/actions/`, rename **createTicketInRT.xml** to **RT-queue1.xml**.

- c Edit **RT-queue1.xml** and change the `<name> . . . </name>` option to something descriptive, like;

```
<name>Create/Update RT-queue1</name>
```

Also update the `--queue` option to `queue1`. Save the file and make similar changes for **RT-queue2.xml**. Make sure to use different `<name> . . . </name>` options.

- 3 By default, a new ticket will be created on a per-test basis. If device A has two tests X and Y, and both tests fail, one ticket for X and one ticket for Y will be created. If you prefer to restrict new tickets to a per-device basis, where information for X and Y will be entered into same ticket (second test information will be added as additional comment), then edit the XML configuration file for the script and add `--perdevice` option to the `<parameters> . . . </parameters>` section.

Configuration on UNIX

- 1 To use the newly added plugin action, you first need to create an action profile that uses this script. Create an action profile via Administration > Actions > **Create New Action** (or update an existing profile). From the Notify Using drop-down list, you should be able to select the script. The name displayed on the list will correspond to the `<name>...</name>` parameter in `$TRAVERSE_HOME/plugin/actions/createTicketInRT.xml` file. The **Message Recipient** field can be left empty and rest of the parameters set as you see fit. Now apply this action profile to various tests as required.
- 2 If you wish to create tickets in different queues, you will need to create two different plugin actions - one each for the two RT queue:

```
su
cd $TRAVERSE_HOME
etc/traverse.init stop
cd plugin/actions
mv createTicketInRT.xml RT-queue1.xml
cp RT-queue1.xml RT-queue2.xml
```

Now edit **RT-queue1.xml** and change the `<name>...</name>` option to something descriptive, like

```
<name>Create/Update RT-queue1</name>
```

Also update the `--queue` option to `queue1`. Save the file and make similar changes for **RT-queue2.xml**. Make sure to use different `<name>...</name>` options.

- 3 By default, a new ticket will be created on a per-test basis. If device A has two tests X and Y, and both tests fail, one ticket for X and one ticket for Y will be created. If you prefer to restrict new tickets to a per-device basis, where information for X and Y will be entered into same ticket (second test information will be added as additional comment), then edit the XML configuration file for the script and add `--perdevice` option to the `<parameters>...</parameters>` section.

Troubleshooting

- 1 Look in `traverse/logs/error.log` for any error messages logged by the DGE process.
- 2 (UNIX) Check to make sure `plugin/actions/createTicketInRT.pl` is executable (mode 0555).

- 3 (UNIX) Try running the script manually to ensure you can create a new ticket:

```
cd /usr/local/traverse/plugin/actions
./createTicketInRT.pl --queue YOUR_QUEUE_NAME \
--rtuser traverse@your.domain \
--device "Sample Device" --test "Sample Test" \
--severity warning --result 100 --unit ms \
--location "Data Center" --type ping/rtt \
--threshold "75/200" --search
```

SECTION 2

Web Services API



This chapter provides an overview of the Traverse Web Services Application Programming Interface (API).

- [About the Traverse Web Services API on page 111](#)
- [Why Use the Traverse Web Services API on page 112](#)
- [Traverse Web Services API Workflow on page 112](#)
- [About the Traverse Web Services on page 113](#)

7.1 About the Traverse Web Services API

The Traverse Web Services API consists of three Web services:

Service	Description
Session Manager	Processes login and logout requests The Federated Security Model in Traverse requires a valid session to be established before any of the other services can be used.
Business Visibility Engine (BVE)	Provides visibility of certain objects on the user network
External Data Feed	

In this release, only a read-only view into Traverse and its objects is available. You cannot create or update Traverse objects.

Depending on login credentials, Traverse exposes a certain set of objects. When using the BVE, you can look up a device and obtain its attributes and status.

You can access the following objects using the Traverse Web Services API:

- Device
- Container
- Data Gathering Engine (DGE)
- Test
- Location
- Subnet
- Department
- User
- Device dependency

7.2 Why Use the Traverse Web Services API

You can use the Traverse Web Services API to create your own portal to provide a limited view of the system (for example, a limited number of devices or an aggregate status showing the health of the network).

7.3 Traverse Web Services API Workflow

The following describes a typical Traverse Web services API workflow:

- 1 Client application uses the Session Manger service to log in to Traverse.
- 2 Traverse sends back a response, including a session identifier (ID).
- 3 The client application uses the session ID to make calls to Traverse through the BVE/EDF service.

The client application uses the session ID with every call.

- 4 The client application ends its interaction with Traverse by logging out using the Session Manager service.

7.4 About the Traverse Web Services

Traverse's Web services support Traverse informational sessions with a remote host.

To log in, you provide one of the following parameters, which are provided by the SEA:

- Username and password
- Session ID and a user type

User Types

The Session Manager service defines three types of users:

- End user
An end user has restricted access to the system. For example, the member of a department.
- SEA
An SEA has limited administrative access to the system.
- Superuser
A superuser has full access to the system.

Traverse WSDL Files

Here is a list of the WSDL API files for accessing the TopologyService Web services:

- `SessionManager.wsdl`: Defines the Session Manager service's operations and types
- `BusinessVisibilityEngine.wsdl`: Defines the Business Visibility Engine service's operations and types

WSDL File Locations

The following is a list of the WSDL file locations:

- Session Manager service
`http://<hostname>/api/SessionManager?wsdl`
- Business Visibility Engine service
`http://<hostname>/api/BusinessVisibilityEngine?wsdl`

where <hostname> is the address of the Traverse host.

7.5 Sample Code

Sample code can be found online in the Zyrion User Community forum at <http://community.zyrion.com/>.

This chapter describes the Session Manager service.

- [Overview on page 115](#)
- [Operations on page 116](#)
- [Types on page 118](#)

8.1 Overview

The Traverse Session Manager service processes login and logout requests.

To log in, you must provide either a previously defined session ID, or a login name and password. To log out, you must provide the same session ID.

Session Types

The Session Manager service supports the following session types:

- Traverse
- SEA

Login Type Parameters

The Session Manager service supports the following login types:

- End user
An end user has restricted access to the system. For example, the member of a department.
- System administrator
A system administrator has limited administrative access to the system.
- Superuser
A superuser has full access to the system.

8.2 Operations

This section describes the following operations of the Session Manager service:

- [login on page 116](#)
- [logout on page 117](#)

login

Provides authentication information about a user to the server, as a username/password combination or using an SEA-provided session ID.

Parameter

`loginRequest LoginRequest`

Response

`loginResponse LoginResponse`

Example

The following Visual Basic code snippet is an example of how to build a `loginRequest` object and call the `login` operation:

```
...  
  
Dim loginRequest As New nvSessionMgr.LoginRequest  
Dim loginResponse As New nvSessionMgr.LoginResponse  
  
loginRequest.loginType = nvSessionMgr.LoginType.USERNAME  
loginRequest.sessionType = nvSessionMgr.SessionType.NETVIGIL  
  
loginRequest.username = "my_user"  
loginRequest.password = "my_password"  
loginRequest.remoteAddress = "Web Service Client Example"  
  
loginResponse = sessionManagerObject.login(loginRequest)
```

...

logout

Logs out the specified session.

Parameter

logoutRequest [LogoutRequest](#)

Response

logoutResponse [LogoutResponse](#)

Example

The following Visual Basic code snippet is an example of how to build a `logoutRequest` object and call the `logout` operation:

...

```
Dim logoutRequest As New nvSessionMgr.LogoutRequest
logoutRequest.sessionID = loginResponse.sessionID
sessionManagerObject.logout(logoutRequest)
```

...

8.3 Types

This section describes the types provided by the Session Manager service:

[Complex Types on page 118](#)

[Simple Types on page 121](#)

Complex Types

This section describes the following complex types provided by the Traverse Session Manager service:

[LoginRequest on page 118](#)

[LoginResponse on page 119](#)

[LogoutRequest on page 120](#)

[LogoutResponse on page 120](#)

LoginRequest

Specifies the information for authenticating a user.

FIELDS

Name	Type	Description
loginType	LoginType	(Optional) Enumerated type specifying type of login: user name, MD5-encrypted username and password, encoded username, or session ID. This field can be NULL.
password	string	(Optional) Password for current login request. This field can be NULL.
remoteAddress	string	(Optional) IP address of the client's workstation or descriptive text of a custom application. This information is logged in an audit message in BVE. This field can be NULL.
representedUserName	string	(Optional) Name of user, if user is being represented by an SEA. This field should not be used when the loginType is USERNAME. This field can be NULL.
sessionID	string	(Optional) Session ID provided by the SEA for login. This field can be NULL.
sessionType	SessionType	(Optional) Enumerated type indicating whether session is of type TRAVERSE (session ID provided by Traverse in the course of normal login) or SEA (session ID provided before login by an SEA). This field can be NULL.
username	string	(Optional only if sessionID is not provided) Username for login request. This field can be NULL.

LoginResponse

Specifies the information for a response to a login request.

FIELDS

Name	Type	Description
sessionId	string	(Optional) The unique ID generated by Traverse for this session. This field can be NULL.
statusCode	int	(Optional) Code indicating whether login was successful (0).
statusMessage	string	(Optional) Status message. Contains explanation of error if <code>statusCode</code> is greater than 0. This field can be NULL.
userType	UserType	(Optional) The type of user logged in. This field can be NULL.

LogoutRequest

Specifies the information needed for a logout request.

FIELDS

Name	Type	Description
loginType	LoginType	(Optional) Enumerated type specifying type of login session from which to log out. This field can be NULL.
remoteAddress	string	(Optional) The IP address of the Traverse server to log out from. This field can be NULL.
sessionId	string	(Optional) The ID of the session to log out from. This field can be NULL.
sessionType	SessionType	(Optional) Enumerated type specifying the session type. This field can be NULL.

LogoutResponse

Specifies the response to a logout request.

FIELDS

Name	Type	Description
statusCode	int	(Optional) Indicates success (0) or failure of logout operation.
statusMessage	string	(Optional) Status message. This field can be NULL.

Simple Types

This section describes the following simple types provided by the Traverse Session Manager service:

[LoginType on page 121](#)

[SessionType on page 121](#)

[UserType on page 122](#)

LoginType

Enumerated type. Specifies the Traverse login options.

FIELDS

Name	Description
USERNAME	Use user name to log in.
MD5	Use MD5 hash function to encode username and password information in the URL. Using the MD5 login option allows you to bypass the initial login page in Traverse by directly encoding the username and password information in the URL and encrypting this information using a shared MD5 hash key. This mechanism allows you to access Traverse via some other portal where you have already been authenticated.
ENCODED_USERNAME	Use encoded user name to log in.
SESSION_ID	Use session ID to log in.

SessionType

Enumerated type. Specifies the type of session to set up.

FIELDS

Name	Description
NETVIGIL	Use NetVigil-provided user ID and password for login and logout.
SEA	Use SEA-provided session ID for login and logout.

UserType

Enumerated type. Specifies the type of user.

FIELDS

Name	Description
NOT_LOGGED_IN_USER	User is not logged in.
END_USER	User is end user.
ADMIN_USER	User is an SEA.
SUPER_USER	User is a super user.

This chapter describes the Business Visibility Engine (BVE) Service of Traverse, and consists of the following subsections:

- [Overview on page 123](#)
- [Operations on page 124](#)
- [Types on page 130](#)

9.1 Overview

The Traverse Business Visibility Engine service provides access to the status and configuration information of various monitored objects in Traverse.

This service correlates the data from multiple Data Gathering Engines (DGEs), allows end users to look at the real-time status of their devices and add new devices and actions using a Web browser.

This service manages Traverse distributed databases and distributed processing while aggregating summary (status) information in real time.

9.2 Operations

This section describes the following operations of the Business Visibility Engine service:

- [getContainerStatus on page 124](#)
- [getDepartmentStatus on page 125](#)
- [getDeviceStatus on page 125](#)
- [getSubnetStatus on page 126](#)
- [getTestResult on page 126](#)
- [getTestStatus on page 126](#)
- [listContainerObject on page 127](#)
- [listDepartmentObject on page 127](#)
- [listDeviceObject on page 127](#)
- [listDgeObject on page 127](#)
- [listLocationObject on page 128](#)
- [listMonitorConfigObject on page 128](#)
- [listSubnetDeviceObject on page 128](#)
- [listSubnetObject on page 129](#)
- [listTestObject on page 129](#)
- [listUserObject on page 129](#)

getContainerStatus

This operation gets the status of one or several containers. If no container is specified, the status of all containers administered by the currently logged-in user is returned.

A container is a personalized view showing the real-time performance of business services, derived from the performance of one's underlying IT infrastructure.

Parameter

`getContainerStatusRequest` [GetStatusRequest](#)

Response

`getContainerStatusResponse` [GetMultiStatusResponse](#)

getDepartmentStatus

This operation gets the status of one or more specified departments. If no department is specified, the status of all departments administered by the currently logged-in user is returned.

A department is a Traverse object that allows users associated with it to manage network devices owned by it.

Parameter

`getDepartmentStatusRequest` [GetStatusRequest](#)

Response

`getDepartmentStatusResponse` [GetMultiStatusResponse](#)

Example

This Java example...

getDeviceStatus

This operation gets the status of one or several devices specified. If no device is specified, the status of all devices administered by the currently logged-in user is returned.

A device is a physical component of the network, such as a router, switch, or hub.

Parameter

`getDeviceStatusRequest` [GetStatusRequest](#)

Response

`getDeviceStatusResponse` [GetMultiStatusResponse](#)

getSubnetStatus

This operation gets the status of one or several subnets. If no subnet is specified, the status of all tests administered by the current user is returned.

A subnet is a range of logical addresses within the address space assigned to an organization.

Parameter

getSubnetStatusRequest [GetStatusRequest](#)

Response

getSubnetStatusResponse [GetMultiStatusResponse](#)

getTestResult

This operation gets the detailed test results for the specified tests. If no test is specified, the test results of all tests administered by the current user is returned.

Parameter

getTestResultRequest [GetStatusRequest](#)

Response

getTestResultResponse [GetTestResultResponse](#)

getTestStatus

This operation gets the status of one or several tests. If no test is specified, the status of all tests administered by the current user is returned.

Parameter

getTestStatusRequest [GetStatusRequest](#)

Response

getTestStatusResponse [GetStatusResponse](#)

listContainerObject

This operation allows you to search for Traverse container objects, based on certain search criteria (filters).

Parameter

`listContainerObject` [ObjectInfoRequest](#)

Response

`listContainerObjectResponse` [ContainerInfoResponse](#)

listDepartmentObject

This operation allows you to search for Traverse department objects based on search criteria (filters).

Parameter

`listDepartmentObject` [ObjectInfoRequest](#)

Response

`listDepartmentObjectResponse` [DepartmentInfoResponse](#)

listDeviceObject

This operation lists Traverse Device objects based on search criteria (filters).

Parameter

`listDeviceObject` [ObjectInfoRequest](#)

Response

`listDeviceObjectResponse` [DeviceInfoResponse](#)

listDgeObject

This operation lists Traverse Data Gathering Engine objects based on search criteria (filters).

Parameter

`listDgeObject` [ObjectInfoRequest](#)

Response

`listDgeObjectResponse` [DgeInfoResponse](#)

listLocationObject

This operation lists Traverse Location objects based on search criteria (filters).

Parameter

`listLocationObject` [ObjectInfoRequest](#)

Response

`ListLocationObjectResponse` [LocationInfoResponse](#)

listMonitorConfigObject

This operation lists Traverse Monitor Config objects based on search criteria.

Parameter

`listMonitorConfigObject` [ObjectInfoRequest](#)

Response

`listMonitorConfigObjectResponse` [MonitorConfigInfoResponse](#)

listSubnetDeviceObject

This operation lists Traverse Subnet Device objects based on search criteria.

Parameter

`listSubnetDeviceObject` [ObjectInfoRequest](#)

Response

`listSubnetDeviceObject` Response [DeviceInfoResponse](#)

listSubnetObject

This operation lists Traverse Subnet objects based on search criteria.

Parameter

listSubnetObject [ObjectInfoRequest](#)

Response

listSubnetObjectResponse [SubnetInfoResponse](#)

listTestObject

This operation lists Traverse Test objects based on search criteria.

Parameter

listTestObject [ObjectInfoRequest](#)

Response

listTestObjectResponse [TestInfoResponse](#)

listUserObject

This operation lists Traverse User objects based on search criteria.

Parameter

listUserObject [ObjectInfoRequest](#)

Response

listUserObjectResponse [UserInfoResponse](#)

9.3 Types

This section describes the following complex types provided by the Traverse Business Visibility Engine Service:

[ArrayOfContainerInfo](#) on page 132

[ArrayOfDepartmentInfo](#) on page 132

[ArrayOfDependencyOperationStatus](#) on page 132

[ArrayOfDeviceDependency](#) on page 133

[ArrayOfDeviceInfo](#) on page 133

[ArrayOfDgeInfo](#) on page 133

[ArrayOfLocationInfo](#) on page 134

[ArrayOfLong](#) on page 134

[ArrayOfMonitorConfigInfo](#) on page 134

[ArrayOfObjectMultiStatus](#) on page 135

[ArrayOfObjectStatus](#) on page 135

[ArrayOfPair](#) on page 135

[ArrayOfString](#) on page 136

[ArrayOfSubnetInfo](#) on page 136

[ArrayOfTestInfo](#) on page 136

[ArrayOfTestResultInfo](#) on page 137

[ArrayOfUserInfo](#) on page 137

[ContainerInfo](#) on page 137

[ContainerInfoResponse](#) on page 139

[DepartmentInfo](#) on page 139

[DepartmentInfoResponse](#) on page 141

[DependencyOperationStatus](#) on page 142

[DeviceDependencyRequest](#) on page 142

[DeviceDependencyResponse](#) on page 143

[DeviceDependency](#) on page 142

[DeviceInfo](#) on page 143

[DeviceInfoResponse](#) on page 146

[DgeInfo](#) on page 147

[DgeInfoResponse](#) on page 147

[GetMultiStatusResponse](#) on page 148

[GetStatusRequest](#) on page 148

[GetStatusResponse](#) on page 149

[GetTestResultResponse](#) on page 149

[LocationInfo](#) on page 150

[LocationInfoResponse](#) on page 150

[MonitorConfigInfo](#) on page 151

[MonitorConfigInfoResponse](#) on page 152

[ObjectInfoRequest](#) on page 153

[ObjectMultiStatus](#) on page 153

[ObjectStatus](#) on page 154

[Pair](#) on page 154

[SubnetInfo](#) on page 155

[SubnetInfoResponse](#) on page 155

[TestInfo](#) on page 156

[TestInfoResponse](#) on page 158

[TestResultInfo](#) on page 159

[UserInfo](#) on page 160

[UserInfoResponse](#) on page 162

[VisualizerIpPointInfo](#) on page 162

ArrayOfContainerInfo

Represents an array of one or more elements of type ContainerInfo in the Traverse database.

FIELDS

Name	Type	Description
ContainerInfo	ContainerInfo	(Optional) Array of one or more elements of type ContainerInfo. This field can be NULL.

ArrayOfDepartmentInfo

Represents an array of one or more elements of type DepartmentInfo in the Traverse database.

FIELDS

Name	Type	Description
DepartmentInfo	DepartmentInfo	(Optional) Array of one or more elements of type DepartmentInfo. This field can be NULL.

ArrayOfDependencyOperationStatus

Represents an array of one or more elements of type DependencyOperationStatus in the Traverse database.

FIELDS

Name	Type	Description
DependencyOperationStatus	DependencyOperationStatus	(Optional) Array of one or more elements of type DependencyOperationStatus. This field can be NULL.

ArrayOfDeviceDependency

Represents an array of one or more elements of type DeviceDependency in the Traverse database.

FIELDS

Name	Type	Description
DeviceDependency	DeviceDependency	(Optional) Array of one or more elements of type DeviceDependency. This field can be NULL.

ArrayOfDeviceInfo

Represents an array of one or more elements of type DeviceInfo in the Traverse database.

FIELDS

Name	Type	Description
DeviceInfo	DeviceInfo	(Optional) Array of one or more elements of type DeviceInfo. This field can be NULL.

ArrayOfDgeInfo

Represents an array of one or more elements of type DgeInfo in the Traverse database.

FIELDS

Name	Type	Description
DgeInfo	DgeInfo	(Optional) Array of one or more elements of type DgeInfo. This field can be NULL.

ArrayOfLocationInfo

Represents an array of one or more elements of type `LocationInfo` in the Traverse database.

FIELDS

Name	Type	Description
LocationInfo	<code>LocationInfo</code>	(Optional) Array of one or more elements of type <code>LocationInfo</code> . This field can be NULL.

ArrayOfLong

Represents an array of one or more elements of type `long`.

FIELDS

Name	Type	Description
ArrayOfLong	<code>long</code>	(Optional) Array of one or more elements of type <code>long</code> . This field can be NULL.

ArrayOfMonitorConfigInfo

Represents an array of one or more elements of type `MonitorConfigInfo` in the Traverse database.

FIELDS

Name	Type	Description
MonitorConfigInfo	<code>MonitorConfigInfo</code>	(Optional) Array of one or more elements of type <code>MonitorConfigInfo</code> . This field can be NULL.

ArrayOfObjectMultiStatus

Represents an array of one or more elements of type `ObjectMultiStatus` in the Traverse database.

FIELDS

Name	Type	Description
<code>ObjectMultiStatus</code>	<code>ObjectMultiStatus</code>	(Optional) Array of one or more elements of type <code>ObjectMultiStatus</code> . This field can be NULL.

ArrayOfObjectStatus

Represents an array of one or more elements of type `ObjectStatus` in the Traverse database.

FIELDS

Name	Type	Description
<code>ObjectStatus</code>	<code>ObjectStatus</code>	(Optional) Array of one or more elements of type <code>ObjectStatus</code> . This field can be NULL.

ArrayOfPair

Represents an array of one or more elements of type `Pair` in the Traverse database.

FIELDS

Name	Type	Description
<code>Pair</code>	<code>Pair</code>	(Optional) Array of one or more elements of type <code>Pair</code> . This field can be NULL.

ArrayOfString

Represents an array of one or more elements of type String in the Traverse database.

FIELDS

Name	Type	Description
String	String	(Optional) Array of one or more elements of type String.
		This field can be NULL.

ArrayOfSubnetInfo

Represents an array of one or more elements of type SubnetInfo in the Traverse database.

FIELDS

Name	Type	Description
SubnetInfo	SubnetInfo	(Optional) Array of one or more elements of type SubnetInfo.
		This field can be NULL.

ArrayOfTestInfo

Represents an array of one or more elements of type TestInfo in the Traverse database.

FIELDS

Name	Type	Description
TestInfo	TestInfo	(Optional) Array of one or more elements of type TestInfo.
		This field can be NULL.

ArrayOfTestResultInfo

Represents an array of one or more elements of type `TestResultInfo` in the Traverse database.

FIELDS

Name	Type	Description
<code>TestResultInfo</code>	<code>TestResultInfo</code>	(Optional) Array of one or more elements of type <code>TestResultInfo</code> . This field can be NULL.

ArrayOfUserInfo

Represents an array of one or more elements of type `UserInfo` in the Traverse database.

FIELDS

Name	Type	Description
<code>UserInfo</code>	<code>UserInfo</code>	(Optional) Array of one or more elements of type <code>UserInfo</code> . This field can be NULL.

ContainerInfo

Specifies information about a container.

FIELDS

Name	Type	Description
childContainer	ArrayOfLong	(Optional) Child containers nested within this container. This field can be NULL.
childDevice	ArrayOfLong	(Optional) Dependent devices in this container. This field can be NULL.
comment	string	(Optional) Administrator comments. This field can be NULL.
creationTime	long	(Optional) When this container was created. The date and time format is yyyy.mm.dd-hh:mm.
department	long	(Optional) Identity of department pertaining to this container
departmentName	string	(Optional) Name of department pertaining to this container. This field can be NULL.
deviceType	int	(Optional) The device type of the device referred to in this container.
name	string	(Optional) Name of the container. This field can be NULL.
parentContainer	ArrayOfLong	(Optional) Parent container in which this container is nested. This field can be NULL.
parentDevice	ArrayOfLong	(Optional) Parent device for which this container is existent. This field can be NULL.
readOnly	boolean	(Optional) Whether this container is read-only (true).
serialNumber	long	(Optional) Serial number for this container.

Name	Type	Description
showNetworkDevicesAsContainers	boolean	(Optional) Whether or not to show the underlying network devices as containers (true) or as a list of tests (false).
test	ArrayOfLong	(Optional) Tests stored in this container if it is used as a virtual device. This field can be NULL.
view	int	(Optional) ID of the container's view.

ContainerInfoResponse

Specifies the response to a listContainerObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfContainerInfo	(Optional) An array of the resulting containers matching the request. This field can be NULL.
statusCode	int	(Optional) The result of the listContainerObject request. A value of 0 indicates success.
statusMessage	string	(Optional) Status message. This field can be NULL.

DepartmentInfo

Specifies information about a department of a service container.

FIELDS

Name	Type	Description
address1	string	(Optional) First line of departmental address. This field can be NULL.
address2	string	(Optional) Second line of departmental address. This field can be NULL.
city	string	(Optional) City of departmental address. This field can be NULL.
company	string	(Optional) Company for this department. This field can be NULL.
contactEmail	string	(Optional) Email address of departmental contact. This field can be NULL.
contactPhone	string	(Optional) Telephone number of departmental contact. This field can be NULL.
country	string	(Optional) Country of departmental address. This field can be NULL.
creationTime	long	(Optional) Creation time of this department.
customerId	string	(Optional) Unique Customer ID for this department. This field can be NULL.
groupId	long	(Optional) Group ID of this department.
groupName	string	(Optional) Group name of the department. This field can be NULL.
groupType	string	(Optional) Group type of this department. This field can be NULL.
hierarchyContainers	ArrayOfLong	(Optional) List of one or more hierarchy containers in this department, if present. This field can be NULL.
lastLoginTime	long	(Optional) Last login time for this department.
name	string	(Optional) Department name. This field can be NULL.

Name	Type	Description
networkDevices	ArrayOfLong	(Optional) List of one or more networked devices in this department, if present. This field can be NULL.
numSessions	int	(Optional) Number of sessions active for this department.
serialNumber	long	(Optional) Serial number of this department?
state	string	(Optional) State of departmental address. This field can be NULL.
subnet	ArrayOfLong	(Optional) List of one or more subnets in this department, if present. This field can be NULL.
suspended	boolean	(Optional) Whether this department is suspended (true).
user	ArrayOfLong	(Optional) List of one or more users in this department, if present. This field can be NULL.
whySuspended	string	(Optional) If this department is suspended, reason for suspension. This field can be NULL.
zip	string	(Optional) Zip-code of departmental address. This field can be NULL.

DepartmentInfoResponse

Specifies the response to a listDepartmentObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfDepartmentInfo	(Optional) The information returned about the departments specified in the request. This field can be NULL.
statusCode	int	(Optional) Success (0) or failure of the request.
statusMessage	string	(Optional) Status message. This field can be NULL.

DependencyOperationStatus

Specifies the status of the dependency verification request operation.

FIELDS

Name	Type	Description
dependency	DeviceDependency	(Optional) The dependency to check. This field can be NULL.
message	string	(Optional) Status message. This field can be NULL.

DeviceDependency

A device dependency *is a parent-child relationship between monitored devices*. A single parent can have multiple children, and a single child can have multiple parents. Device dependencies are cascading. If A is a child of B, and B is a child of C, it is only necessary to configure A as a child of B and B as a child of C. The Business Visibility Engine automatically recognizes the dependency between A and C.

If a device is tested and the result is *critical* (for all thresholds), *unknown*, or *failed*, some additional processing is used to determine if the device is reachable. If the Data Gathering Engine cannot access any of the child's parent devices, the child is considered *unreachable*.

FIELDS

Name	Type	Description
childId	long	(Optional) Child device having a dependency on this device. This field can be NULL.
parentId	long	(Optional) Parent device on which this device has a dependency. This field can be NULL.

DeviceDependencyRequest

Specifies a list of dependencies to check.

FIELDS

Name	Type	Description
dependency	ArrayOfDeviceDependency	(Optional) Specifies the dependencies to check. This field can be NULL.
sessionId	string	(Optional) For authentication purposes, specifies the session in which the request is made. This field can be NULL.

DeviceDependencyResponse

Specifies the result of a dependency discovery operation.

FIELDS

Name	Type	Description
results	ArrayOfDependencyOperationStatus	(Optional) Specifies the dependency status per device checked. This field can be NULL.
statusCode	int	(Optional) Specifies whether the dependency discovery request succeeded (0) or failed.
statusMessage	string	(Optional) Status message. This field can be NULL.

DeviceInfo

Specifies the information about a device (typically discovered during network autodiscovery phase).

FIELDS

Name	Type	Description
address	string	(Optional) The fully qualified domain name (FQDN) or IP address for this device. This field can be NULL.
childContainer	ArrayOfLong	(Optional) One or more containers nested in this device. This field can be NULL.
childDevice	ArrayOfLong	(Optional) One or more devices dependent on this device. This field can be NULL.
childSubnet	ArrayOfLong	(Optional) One or more subnets dependent on this device. This field can be NULL.
comment	string	(Optional) Administrator comments. This field can be NULL.
creationTime	long	(Optional) Creation time of this device.
department	long	(Optional) ID of department owning this device.
departmentName	string	(Optional) Name of department owning this device. This field can be NULL.
deviceType	int	(Optional) Device type ID.
deviceTypeString	string	(Optional) Device type. This field can be NULL.
dge	long	(Optional) Data gathering engine ID for this device.
ipAddress	string	(Optional) IP address for this device. This field can be NULL.
location	long	(Optional) Location of this device.
model	string	(Optional) Device model. This field can be NULL.

Name	Type	Description
monitorConfig	ArrayOfLong	(Optional) Monitor configuration for this device. This field can be NULL.
name	string	(Optional) The name of this device. This field can be NULL.
parentContainer	ArrayOfLong	(Optional) One or more containers within which this device is referenced. This field can be NULL.
parentDevice	ArrayOfLong	(Optional) One or more devices this device depends on. This field can be NULL.
readOnly	boolean	(Optional) Whether this device can only be read from (true).
resolvedAddress	ArrayOfString	(Optional) One or more valid IP addresses for this network device. This field can be NULL.
serialNumber	long	(Optional) Serial number of this device.
smartNotify	boolean	(Optional) Whether this is a Smart Notify and Alarm Protocol (SNAP) device (true).
suspended	boolean	(Optional) Whether this device is suspended (true).
tag1	string	(Optional) Space for user-defined parameter. This field can be NULL.
tag2	string	(Optional) Space for user-defined parameter. This field can be NULL.
tag3	string	(Optional) Space for user-defined parameter. This field can be NULL.
tag4	string	(Optional) Space for user-defined parameter. This field can be NULL.

Name	Type	Description
tag5	string	(Optional) Space for user-defined parameter. This field can be NULL.
test	ArrayOfLong	(Optional) Tests pertaining to this device. This field can be NULL.
type	string	(Optional) Device type. This field can be NULL.
vendor	string	(Optional) Device vendor. This field can be NULL.
view	int	(Optional) ID of the device's view.
visIPoint	VisualizerIpoint Info	(Optional) The Visualizer Instrumentation Point for this device. This field can be NULL.
vizAdvancedReportUrl	string	(Optional) URL where Visualizer Advanced Report is stored. This field can be NULL.
vizDashboardUrl	string	(Optional) URL where Visualizer Dashboard should be placed. This field can be NULL.
vizQuickReportUrl	string	(Optional) URL where Visualizer Quick Report is stored. This field can be NULL.

DeviceInfoResponse

Specifies the response to a listDeviceObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfDeviceInfo	(Optional) The information returned about the individual devices. This field can be NULL.

Name	Type	Description
statusCode	int	(Optional) Response status (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.

DgeInfo

Specifies the information about a DGE.

FIELDS

Name	Type	Description
creationTime	long	(Optional) Time at which the DGE was created.
hardLimit	int	(Optional) The number of tests beyond which no more tests can be provisioned on that DGE.
host	string	(Optional) DGE host. This field can be NULL.
location	long	(Optional) DGE location.
name	string	(Optional) DGE name. This field can be NULL.
serialNumber	long	(Optional) DGE serial number.
softLimit	int	(Optional) Once a soft limit is reached, tests for only existing devices can be added to that DGE. Else the device is provisioned on the least loaded DGE.
testCount	int	(Optional) Number of tests.

DgeInfoResponse

Specifies the Traverse response to a listDgeObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfDgeInfo	(Optional) Information about DGEs. This field can be NULL.
statusCode	int	(Optional) Result of operation (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.

GetMultiStatusResponse

Specifies the Traverse response to a listMultiObject request.

FIELDS

Name	Type	Description
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.
statuses	ArrayOfObjectMultiStatus	(Optional) A list of objects and their status. This field can be NULL.

GetStatusRequest

Requests the status of one or more objects.

FIELDS

Name	Type	Description
serialNumber	ArrayOfLong	(Optional) The serial numbers of the objects for which status is requested. This field can be NULL.
sessionId	string	(Optional) Session ID (for authentication purposes) required when making a request. This field can be NULL.

GetStatusResponse

Specifies the response to a status request. If successful, specifies a list of objects and their status.

FIELDS

Name	Type	Description
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.
statuses	ArrayOfObjectStatus	(Optional) List of objects and their status. This field can be NULL.

GetTestResultResponse

Specifies Traverse response to a listTestObject request.

FIELDS

Name	Type	Description
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.
testResult	ArrayOfTestResultInfo	(Optional) One or more test results. This field can be NULL.

LocationInfo

Specifies the information about a Traverse location object.

FIELDS

Name	Type	Description
city	string	(Optional) The City for this location. This field can be NULL.
comments	string	(Optional) Administrator comments. This field can be NULL.
creationTime	long	(Optional) Creation time of the location object.
dge	long	(Optional) DGE, if present, for this location. This field can be NULL.
name	string	(Optional) Name for this location. This field can be NULL.
serialNumber	long	(Optional) Serial number for this location object
state	string	(Optional) The geographical state for this location object. This field can be NULL.
streetAddress	string	(Optional) The street address for this location object. This field can be NULL.

LocationInfoResponse

Specifies the response to a listLocationObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfLocationInfo	(Optional) Location info returned by the query. This field can be NULL.
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.

MonitorConfigInfo

Specifies information about a Traverse Plugin Monitor configuration.

FIELDS

Name	Type	Description
childContainer	ArrayOfLong	(Optional) List of child containers. This field can be NULL.
childDevice	ArrayOfLong	(Optional) List of child devices. This field can be NULL.
comment	string	(Optional) SEA comments about this configuration. This field can be NULL.
creationTime	long	(Optional) Time at which the information was created.
department	long	(Optional) Department ID.
departmentName	string	(Optional) Department name. This field can be NULL.
deviceType	int	(Optional) Device type.
monitorType	string	(Optional) The monitor type of this configuration. This field can be NULL.
name	string	(Optional) Configuration name. This field can be NULL.
parentContainer	ArrayOfLong	(Optional) List of parent container IDs. This field can be NULL.
parentDevice	ArrayOfLong	(Optional) List of parent device IDs. This field can be NULL.
properties	ArrayOfPair	(Optional) List of properties. This field can be NULL.
readOnly	boolean	(Optional) Whether this configuration is read-only (true).
serialNumber	long	(Optional) Configuration serial number.
test	ArrayOfLong	(Optional) List of test IDs. This field can be NULL.
view	int	(Optional) ID of the container view.

MonitorConfigInfoResponse

Specifies the response to a listMonitorConfigObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfMonitorConfigInfo	(Optional) Information about requested monitor configuration. This field can be NULL.
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.

ObjectInfoRequest

Specifies the request for information about a Traverse object (such as a container, department, device, DGE, location, monitor configuration, subnet, test, or user).

FIELDS

Name	Type	Description
serialNumbers	ArrayOfLong	(Optional) A list of object serial numbers. This field can be NULL.
sessionId	string	(Optional) Session ID, required for authentication. This field can be NULL.

ObjectMultiStatus

Specifies the various statuses pertaining to a Traverse object.

FIELDS

Name	Type	Description
applicationStatus	int	(Optional) Status of application.
networkStatus	int	(Optional) Status of network.
serialNumber	long	(Optional) Serial number of object.
status	int	(Optional) Status of object.
statusText	string	(Optional) Status text. This field can be NULL
systemStatus	int	(Optional) Status of system.

ObjectStatus

Contains the status of a Traverse object.

FIELDS

Name	Type	Description
serialNumber	long	(Optional) Serial number of the object.
status	int	(Optional) Status of the object.
statusText	string	(Optional) Status text. This field can be NULL

Pair

Specifies a key/value pair.

FIELDS

Name	Type	Description
key	string	(Optional) Name of the key. This field may be NULL.
value	string	(Optional) Value of the key. This field may be NULL.

SubnetInfo

Specifies information about a Traverse Subnet object.

FIELDS

Name	Type	Description
creationTime	long	(Optional) Subnet creation time.
department	long	(Optional) Department the subnet belongs to.
departmentName	string	(Optional) Name of department the subnet belongs to.
gateway	ArrayOfLong	(Optional) List of subnet gateways. This field may be NULL.
name	string	(Optional) Subnet name. This field may be NULL.
networkAddress	string	(Optional) Network address for this subnet. This field may be NULL.
serialNumber	long	(Optional) Subnet's serial number.
subnetMask	string	(Optional) Subnet mask. This field may be NULL.

SubnetInfoResponse

Specifies the response to a listSubnetObject request.

FIELDS

Name	Type	Description
<code>objectInfo</code>	<code>ArrayOfSubnetInfo</code>	(Optional) Information about subnets returned in response to query. This field may be NULL.
<code>statusCode</code>	<code>int</code>	(Optional) Operation result (0 indicates success).
<code>statusMessage</code>	<code>string</code>	(Optional) Status message. This field can be NULL.

TestInfo

Specifies information about a Traverse Test object.

FIELDS

Name	Type	Description
comment	string	(Optional) SEA comments. This field may be NULL.
creationTime	long	(Optional) Test creation time.
interval	int	(Optional) How often the test is to be run.
name	string	(Optional) Test name. This field may be NULL.
networkDevice	long	(Optional) Network device to run the test on.
serialNumber	long	(Optional) Test's serial number.
shadowCriticalThresh hold	string	(Optional) The default admin critical threshold for this test type. This field may be NULL.
shadowWarningThresh old	string	(Optional) The default admin warning threshold for this test type. This field may be NULL.
slaThreshold	string	(Optional) The default SLA threshold for this test type. This field may be NULL.
suppressed	boolean	(Optional) True if test is suppressed.
suspended	boolean	(Optional) True if test is suspended.
testSubType	string	(Optional) Test subtype of test. This field may be NULL.
testType	string	(Optional) Test type. This field may be NULL.
thresholdType	int	(Optional) Test threshold type.
units	string	(Optional) The units for the test measurement. This will be used in reports and event and summary displays. If the particular test does not have a suitable unit, use a space as the unit. This field may be NULL.

Name	Type	Description
userCriticalThreshold	string	(Optional) Critical threshold set by the user for this test. This field may be NULL.
userWarningThreshold	string	(Optional) Warning threshold set by the user for this test. This field may be NULL.

TestInfoResponse

Specifies the response to a listTestObject request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfTestInfo	(Optional) Test information returned in response to the query. This field can be NULL.
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.

TestResult

Specifies the result of a test.

FIELDS

Name	Type	Description
timestamp	long	Test time.
value	int	Test result.
userWarningThreshold	int	Warning threshold set by user.
userCriticalThreshold	int	Critical threshold set by user.

Name	Type	Description
displayCategory	string	(Optional) The display category. Valid values are network, system, and application. This field can be NULL.
timeEnteredCurrentState	long	Time at which the device being tested entered the current state.
errorMessage	string	Error message generated by the test.

TestResultInfo

Specifies information about a Traverse TestResult object.

FIELDS

Name	Type	Description
displayCategory	string	(Optional) The display category. Valid values are network, system, and application. This field can be NULL.
errorMessage	string	(Optional) Error message generated by test. This field can be NULL.
serialNumber	long	(Optional) The serial number of the test.
severity	int	(Optional) The severity of test result errors.
timeEnteredCurrentState	long	(Optional) Time at which the device being tested entered the current state.
timestamp	long	(Optional) Time when test was performed.
userCriticalThreshold	string	(Optional) Critical test threshold level set by user. This field can be NULL.
userWarningThreshold	string	(Optional) Warning threshold set by user. This field can be NULL.
value	int	(Optional) Test result.

TestResultResponse

Specifies the response to a `getTestResult` request.

FIELDS

Name	Type	Description
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.
testResult	ArrayOfTestResultInfo	(Optional) Test results. This field can be NULL.

UserInfo

Specifies information about a Traverse User object.

FIELDS

Name	Type	Description
comment	string	(Optional) SEA comments about user. This field can be NULL.
creationTime	long	(Optional) When the user was created.
department	long	(Optional) User's department.
departmentName	string	(Optional) Department name. This field can be NULL.
emailAddresses	string	(Optional) User's email address. This field can be NULL.
firstName	string	(Optional) User's first name. This field can be NULL.
lastLoginTime	long	(Optional) Time the user logged in last.
lastName	string	(Optional) User's last name. This field can be NULL.
localeCountry	string	(Optional) User's country locale. This field can be NULL.
localeLanguage	string	(Optional) User's language locale. This field can be NULL.
loginName	string	(Optional) User's login name. This field can be NULL.
numSessions	int	(Optional) Number of user's currently active sessions.
pager	string	(Optional) User's pager number. This field can be NULL.
phoneDay	string	(Optional) User's daytime phone number. This field can be NULL.
phoneEvening	string	(Optional) User's evening phone number. This field can be NULL.
phoneMobile	string	(Optional) User's mobile phone number. This field can be NULL.
role	string	(Optional) User's role in the department. This field can be NULL.

Name	Type	Description
serialNumber	long	(Optional) User's serial number.
timeZone	string	(Optional) User's time zone. This field can be NULL.

UserInfoResponse

Specifies the response to listUserInfo request.

FIELDS

Name	Type	Description
objectInfo	ArrayOfUserInfo	(Optional) Array of type UserInfo returned by the query. This field can be NULL.
statusCode	int	(Optional) Operation result (0 indicates success).
statusMessage	string	(Optional) Status message. This field can be NULL.

VisualizerIpPointInfo

Specifies information about a Visualizer instrumentation point.

FIELDS

Name	Type	Description
IPointId	string	(Optional) The ID for this instrumentation point. This field can be NULL.
IPointType	int	(Optional) Type of instrumentation point (Interface-only, Interface with subnet, or Interface with Virtual Circuit).
active	boolean	(Optional) Whether the instrumentation point is active (true).
bandwidth	long	(Optional) The bandwidth used by the instrumentation point.

Name	Type	Description
cost	double	(Optional) The cost associated with this instrumentation point.
creationTime	long	(Optional) When this instrumentation point was created.
externalAddress	string	(Optional) The external address for this instrumentation point. This field can be NULL.
interfaceId	int	(Optional) The unique ID for this interface.
interfaceIsGroup	boolean	(Optional) Whether this interface is a group (true).
interfaceName	string	(Optional) The name of this interface. This field can be NULL.
mediaType	string	(Optional) Media type for this instrumentation point. The possible values are: <ul style="list-style-type: none"> Ethernet WAN ATM This field can be NULL.
port	int	(Optional) The port for this instrumentation point
protocol	string	(Optional) The protocol used by this instrumentation point. This field can be NULL.
serialNumber	long	(Optional) The serial number of this instrumentation point.
serverIPAddress	string	(Optional) The server's IP address. This field can be NULL.
vcId	long	(Optional) ID of Virtual Circuit. This field uniquely identifies a virtual circuit in the Visualizer database.
vcIsGroup	boolean	(Optional) Whether a virtual circuit is a group (true).

Name	Type	Description
vcName	string	(Optional) Name of virtual circuit. This field can be NULL.
vcType	int	(Optional) Virtual circuit type.
visualizerName	string	(Optional) Name of the Visualizer instance being accessed. This field can be NULL.

This chapter describes the External Data Feed (EDF) service.

This chapter consists of the following subsections:

- [Overview on page 165](#)
- [Operations on page 165](#)
- [Complex Types on page 166](#)

10.1 Overview

The EDF service allows external data to be sent to and processed by Traverse as though it had been collected by Traverse itself. Any external tool can send results and events for any existing test, and the result/event will be processed as if a Traverse monitor had polled the result.

10.2 Operations

This section describes the following operations of the EDF Service:

- [insertTestResults on page 165](#)

insertTestResults

Inserts test results without having to run a test inside a Data Gathering Engine (DGE).

Parameter

```
insertTestResults ResultInsertRequest
```

Response

```
insertTestResultsResponse ResultInsertResponse
```

10.3 Complex Types

This section describes the following complex types provided by the Traverse EDF service:

[ArrayOfBaseResponse](#) on page 166

[ArrayOfTestResult](#) on page 166

[BaseResponse](#) on page 166

[ResultInsertRequest](#) on page 167

[ResultInsertResponse](#) on page 167

[TestResult](#) on page 168

ArrayOfBaseResponse

A list of status codes and status messages describing the result of tests.

FIELDS

Name	Type	Description
BaseResponse	BaseResponse	(Optional) A test's status code and status message. This field can repeat. This field can be NULL.

ArrayOfTestResult

Specifies a list of test results.

FIELDS

Name	Type	Description
TestResult	TestResult	(Optional) Test result information. This field can repeat. This field can be NULL.

BaseResponse

Specifies whether a test succeeded and provides an explanation if the test failed.

FIELDS

Name	Type	Description
<code>statusCode</code>	<code>int</code>	(Optional) Code indicating whether test was successful (0).
<code>statusMessage</code>	<code>string</code>	(Optional) Explanation of error if <code>statusCode</code> is greater than 0. This field can be NULL.

ResultInsertRequest

Specifies the list of test results to insert into Traverse.

FIELDS

Name	Type	Description
<code>results</code>	<code>ArrayOfTestResult</code>	(Optional) List of test results to insert. This field can be NULL.
<code>sessionId</code>	<code>string</code>	(Optional) For authentication purposes, specifies the session in which the request is made. This is the session ID you obtained from Traverse after a successful login request using the <code>login</code> operation. This field can be NULL.

ResultInsertResponse

Specifies whether the `insertTestResults` operation succeeded. Also specifies a list of base responses to the inserted test results specified in the `ResultInsertRequest` parameter.

FIELDS

Name	Type	Description
statusCode	int	(Optional) Code indicating whether the <code>insertTestResults</code> operation was successful (0).
statusMessage	string	(Optional) Explanation of error if <code>statusCode</code> is greater than 0. This field can be NULL.
statuses	ArrayOfBaseResponse	(Optional) List of base responses to the inserting of test results. This field can be NULL.

TestResult

Describes the results of a test against applications, databases, network equipment, or servers.

FIELDS

Name	Type	Description
dateTime	string	(Optional) Date and time of test. The date and time are provided in yyyy.mm.dd-hh:mm format. This field can be NULL.
deviceAddress	string	(Optional) The IP address of the device. This field can be NULL.
deviceName	string	(Optional) The descriptive name used when the device was provisioned. This field can be NULL.
rawResultValue	string	(Optional) The value inserted into the database. The provided result will be multiplied by the result multiplier, and processed in the manner set via <code>process-directive</code> , both set during the creation of the test. This field can be NULL.
reason	string	(Optional) The reason for a failed test. This field can be NULL.

Name	Type	Description
<code>testName</code>	<code>string</code>	(Optional) Name of test. This field can be NULL. Name of Test.
<code>testSerialNum ber</code>	<code>long</code>	(Optional) Unique serial number of the test.
