Kaseya

Kaseya 2

Traverse Developers Guide

version R8

September 22, 2014

Agreement

The purchase and use of all Software and Services is subject to the Agreement as defined in Kaseya's "Click-Accept" EULATOS as updated from time to time by Kaseya at http://www.kaseya.com/legal.aspx. If Customer does not agree with the Agreement, please do not install, use or purchase any Software and Services from Kaseya as continued use of the Software or Services indicates Customer's acceptance of the Agreement."

Contents

Preface

About this Guide

This reference guide describes the Plug-In framework, Application Programming Interface (API), and Web Services interface for Kaseya **Traverse**.

Audience

This guide is intended for administrators and 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 **Traverse**, refer to the following documents:

- **Traverse User Guide** *(http://help.kaseya.com/webhelp/EN/tv/R8/EN_traverse_R8.pdf#zoom=70&navpanes=0)*
- **Traverse Release Notes** *(http://help.kaseya.com/webhelp/EN/RN/index.asp#TraverseReleaseNotes.htm)*

Contacting Kaseya

- Customer Support - You can contact Kaseya technical support online at:
  - **https://helpdesk.kaseya.com/home** *(https://helpdesk.kaseya.com/home)*
- Community Resources - You can also visit the following community resources for Kaseya **Traverse**:
  - Knowledge base at: **http://community.kaseya.com/kb/w/wiki/1206.kaseya-traverse.aspx** *(https://helpdesk.kaseya.com/forums/22931123)*
  - Forum at: **http://community.kaseya.com/xsp/f/340.aspx** *(http://community.kaseya.com/xsp/f/340.aspx)*

C h a p t e r 1

BVE FlexAPI Protocol Reference

In This Chapter

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** *(page 44)*.

Connecting to the Server

Communication with the BVE Server consists of two phases: a connection establishment phase and a command-execution phase. After connecting to the TCP port (default 7661) on the BVE server, the remote client authenticates using a username and password, the same as logging in using the web user interface. Once the user is authenticated, all subsequent commands sent to the server are executed with the permissions and privileges of the specified department that the user belongs to. It is possible to change the privilege level at any point in the command-reply phase by entering new authentication information using the login command.

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

The client application establishes a connection to the BVE server by opening a TCP/IP socket on a pre-defined port number specified in `etc/emerald.xml`. The default port number is 7661. Upon establishment of the TCP session, the server greets the client with a welcome message **following the rules outlined below** *(page 4)*.

Disconnecting from the Server

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

Disconnect on Timeout

Also if the BVE server does not receive anything from the client for an extended period, the session times out and disconnects the client from the BVE server. The default timeout is currently 5 minutes.

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 an 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 part of the argument value.
- Double-quotes are not permitted as part of the value.
- For each client command, the server responds 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 responds with a 203 response code (see below). Each set of output is terminated with a newline (`\n`) and the end of the array is 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 `<regexp>`, 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 values should be specified in `YYYYMMDDhhmm` format and in the logged in user's timezone.
- 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`.
- A special parameter `userName=<value>` is available to admin users. When used, the command is executed for the matching objects as the specified user.

Server Response Format

The server always responds to client initiated commands/requests with text in the format: `<status code> <response code> <optional informative text>`
where `<status_code>` is one of the following:

- `OK`, which indicates the command/request was successful
- `ERR`, which indicates failure to execute the request
- and `<response_code>` is a three digit numeric code which provides additional details about the status code.

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

Client Commands

Login

Provides authentication information to the server. Accepts forward slash or space as separator:

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

If you are logged into the BVE server as an administrator, you can perform an operation as another user by specifying the `username=<value>` in commands or by calling `user.represent` to represent another user.

Logout | Quit

Ends a login session.

```
LOGOUT
```

Help

Typing `HELP` will list all the commands available in the API help.

```
HELP <command>
HELP device.list
```

action.x

action.create

Creates a new action. Using `0` for `<n_tests>` on `notifyAfter` or repeat parameters sends an immediate notification and does not repeat the notification respectively. It is possible to get notifications on multiple states by specifying different state names separated by the `|` symbol for the `notifyOn` parameter. Assigning a method of `none` deletes 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>"]
[, "username=<value>"]
```

The `username` parameter can be used by an administrator to create actions for an end-user department.

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>"]
```

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 the 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 the `action.list` command) first, and then the updated parameters for method2. Otherwise action item #1 is removed.

This command requires all the parameters to be specified even if you want to change only some parameters due to the nature of this command.

```
action.update <"actionName=<regexp>" | "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>"]
```

adminClass.x

adminClass.create

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

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

adminClass.delete

Deletes an existing administrative group.

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

adminClass.list

Lists administrative group information based on search criteria.

```
adminClass.list ["groupName=<regexp>" | "adminClassSerial=<value>"]
```

adminClass.update

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

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

adminGroup.x

The `adminGroup` commands are similar in syntax to the `department.x` commands. Please see the description for **department** *(page 14)* commands.

container.x

container.create

This command 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>"]
[, "departmentName=<string>"]
```

The container name must be unique within the end user department or administrative 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}` include the following:
, `memberList=<regexp_1,regexp_2,...>`.

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

Examples:

```
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}`include 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}` include 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"
```

An administrator can use the `departmentName` parameter to create a container in a specific department.

container.delete

Deletes a container.

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

If you are deleting a device container that has other containers nested below it, the 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 specified search criteria.

```
container.list <"serviceName=<regexp>" | "serialNumber=<value>">
[, "serviceType=<device|test>"]
[, "memberListMethod=<auto|manual>"]
[, "severityMethod=<auto|manual>"]
[, "parentNames=none|<value1,value2,...>"]
[, "actionName=none|<value>"]
[, "departmentName=<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 filter by `ruleDeviceName` when `serviceType=device` and `memberListMethod=auto`.
The output includes all parameters from the `container.update` command, with the exception of `newServiceName` and `memberList`.

Examples:

Search for a container by serial number:

```
container.list "serialNumber=90027"
```

```
OK 203 request accepted, records returned: 1
```

```
"serviceName=Critical Servers", "serialNumber=90027",
"serviceType=device", "memberListMethod=manual", "memberListCount=3",
"severityMethod=auto", "parentSerialNumber=", "actionName=none",
"comment=", "displayComment=false"
```

Search for test containers with a specific action profile:

```
container.list "serviceName=*", "serviceType=test", "actionName=Notify
Admin"
```

```
OK 203 request accepted, records returned: 1
```

```
"serviceName=[VoIP] Infrastructure", "serialNumber=200019",
"serviceType=test", "memberListCount=26", "severityMethod=auto",
"parentSerialNumber=130168", "actionName=Notify Admin", "comment=",
"displayComment=false"
```

Search for containers by name/wildcard:

```
container.list "serviceName=*server*"
```

```
OK 203 request accepted, records returned: 3
```

```
"serviceName=Unix Servers", "serialNumber=90016", "serviceType=device",
"memberListMethod=auto", "ruleDeviceName=", "ruleDeviceType=*Unix*",
"ruleDeviceModel=", "ruleDeviceVendor=", "ruleDeviceTag1=",
"ruleDeviceTag2=", "ruleDeviceTag3=", "ruleDeviceTag4=",
"ruleDeviceTag5=", "memberListCount=1", "severityMethod=auto",
"parentSerialNumber=90027", "actionName=none", "comment=",
"displayComment=false"
"serviceName=Critical Servers", "serialNumber=90027",
"serviceType=device", "memberListMethod=manual", "memberListCount=3",
"severityMethod=auto", "parentSerialNumber=", "actionName=none",
"comment=", "displayComment=false"
"serviceName=Windows Servers", "serialNumber=420035",
"serviceType=device", "memberListMethod=manual", "memberListCount=5",
"severityMethod=auto", "parentSerialNumber=90027", "actionName=none",
"comment=", "displayComment=false"
```

Show all top level containers:

```
container.list "serviceName=*", "parentNames=none"
```

```
OK 203 request accepted, records returned: 1
```

```
"serviceName=All Devices", "serialNumber=300003", "serviceType=device",
"memberListMethod=manual", "memberListCount=1", "severityMethod=auto",
"parentSerialNumber=", "actionName=none", "comment=",
"displayComment=true"
```

container.members

Lists the members of a container.

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

Examples:

Show container members by name:

```
container.members "serviceName=All Switches"
```

```
OK 203 request accepted, records returned: 1
```

```
"serviceName=All Switches", "serialNumber=70011", "memberType=device",
"memberName=switch0.zyrion.com", "memberStatus=Critical",
"deviceName=switch0.zyrion.com", "deviceSerialNumber=140000",
"accountName=My_Company", "accountSerialNumber=49"
```

Show container members by serial number:

```
container.members "serialNumber=210186"
```

```
OK 203 request accepted, records returned: 4
```

```
"serviceName=Service Availability", "serialNumber=210186",
"memberType=test", "memberName=Packet Loss", "memberStatus=Unknown",
"deviceName=Email Relay", "accountName=Network General"
"serviceName=Service Availability", "serialNumber=210186",
"memberType=test", "memberName=Round Trip Time", "memberStatus=Unknown",
"deviceName=Email Relay", "accountName=Network General"
"serviceName=Service Availability", "serialNumber=210186",
"memberType=test", "memberName=Packet Loss", "memberStatus=Unknown",
"deviceName=Exchange Server (Frontend)", "accountName=Network General"
"serviceName=Service Availability", "serialNumber=210186",
"memberType=test", "memberName=Round Trip Time", "memberStatus=Unknown",
"deviceName=Exchange Server (Frontend)", "accountName=Network General"
```

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>"]
[, "departmentName=<value>"]
```

container.update

Updates a container.

```
container.update <"serviceName=<regexp>" | "serialNumber=<value>">
[, "newServiceName=<value>"]
[, "serviceType=<device|test>"]
[, "memberListMethod=<auto|manual>"]
[, "memberList=[+,]<regexp_1,regexp_2,...>"]
[, "memberlistAppend=<true | false>"]
[, "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>"]
[, "departmentName=<value>"]
```

In the above parameters:

`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.

The default value of `memberListAppend` is true to prevent accidental deletion of members from a container. If this value is set to false, all previous members of the container will be removed and only the specified members in this command will be part of the container.

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
  +-DC2
        +-DC3
              +-TC1
```

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

This results in a new hierarchy:

```
DC1
  +-DC2
        +-TC1
              +-TC2
```

Example of adding a new member to an existing container:

```
container.update "serviceName=All Devices", "serviceType=device",
"memberListMethod=manual", "memberList=#All RTT Tests"
```

Example of adding a JDBC test to a container:

```
telnet localhost 7661
```

```
login <user> <pass>
```

```
container.update "serviceName=JMX
Stats","servicetype=test","memberlistmethod=manual","testlistmethod=manual
","testName=JDBC Pool (path=/server, name=jdbc/DataSource) Active
Connections","memberlist=host.FQDN","memberlistappend=true"
```

department.x

department.create

Creates new department information. A user login of the same name as the newly created department is also 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.delete

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

```
department.delete
<"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>"]
```

department.resume

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

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

department.suspend

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

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

department.update

Updates information for an existing department.

```
department.update
<"departmentName=<regexp>" | "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>"]
```

device.x

device.create

Creates a new device configuration in the configuration database.

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 are used. These are configured using the Administration > Other > **Test Parameter Discovery** *(http://help.kaseya.com/webhelp/EN/TV/R8/index.asp#17625.htm)* page in the web application. If `rediscoveryEnabled=true`, you must configure the remaining rediscovery parameters. Specify the rediscovery frequency in minutes. 720 (or 12 hours) is the minimum value accepted by the system.

```
device.create "deviceName=<new_value>"
, "address=<new_value>"
, "locationName=<new_value>"
, "deviceType=<nt|windows|unix|linux|solaris|vmware|xen|hyperv
san|nas|storage|router|switch|firewall|slb|proxy|vpnc|printer|wireless|oth
er>"
, "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|suspendAndLog|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.delete

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

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

device.export

Exports tests from one or more devices.

The `device.export` command is available only when logged into the BVE API server as an admin user or `superuser`.

If the `testName` parameter is used, tests with names matching the specified regular expression are exported. Otherwise, all tests from the specified devices are exported.

If there is name conflict, a unique name in the target department is generated. For example, if device X from department P is exported to department Q, but there is already a device named X in Q, the exported device is named `X_imp_<timestamp>`. If X already exists in Q and is based on an exported device from P, then the list of tests in device X in department Q is updated. If a single device is specified, the `newDeviceName` parameter can be used to set the device name in the target department. In this case, that device name must not already exist in the target department.

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

Examples:

```
device.export "deviceName=My_Dev1", "accountName=My_Dept",
"newAccountName=Your_Dept", "newDeviceName=Your_Dev1"
```

```
device.export "deviceName=Router*", "testName=Packet Loss",
"accountName=My_Dept", "newAccountName=Your_Dept"
```

device.list

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

```
device.list
["deviceName=<regexp>" | "deviceSerial=<value>"]
[, "tagN=<regexp>" ]
[, "parentNames=<value1,value2,...>" ]
[, "showLicenseCategory=<true|false>" ]
[,
"licenseCategory=<ping|simple|network|enterprise|appserver|shypervisor|mhy
pervisor|lhypervisor|xlhypervisor|hypervisor|configmgmt>" ]
```

The output of `device.list` shows the current configuration settings for one or more devices. If any search criteria are provided, devices matching the value are included in the output. If the `showLicenseCategory` parameter is specified, the device's license classification is included as well. By default this information is omitted due to the additional queries that need to be performed to determine this information.

Example:

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

device.move

Moves one or more existing devices and all corresponding tests from one department to another.

You must specify the source and destination departments. If a device with the same name already exists in the destination department, the device being moved is renamed to a default obvious name in the destination department.

```
device.move "deviceName=<regexp>"
, "fromDepartmentName=<value>"
, "toDepartmentName=<value>"
```

device.resume

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

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

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

device.suspend

Suspends one or more existing devices and all corresponding tests.

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

device.update

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

To change the IP address of a host, you must specify the new IP address in the `newaddress` parameter. An error is 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.

```
device.update <"deviceName=<regexp>" | "deviceSerial=<value>" |
[, "newaddress=<ip_addr>"]
[, "snmpCid=<new_value>"]
[, "comment=<new_value>"]
[,
"deviceType=<nt|windows|unix|linux|solaris|vmware|xen|hyperv|router|switch
|firewall|slb|proxy|san|nas|vpnc|printer|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>]
```

Examples*:*

To change the name of a device, you have to use the `deviceSerial` as the match, so first find the serial number.

```
device.List "deviceName=[MyOldDevice]"
device.update "deviceSerial=12345" "deviceName=myNewName"
```

To change the IP address, you should set the `newAddress` value, not the IPaddress.

This following example changes the `rediscoveryUpdatedTestsAction` and `rediscoveryFrequency` for devices that have rediscovery enabled using in the web application. The other fields remain unchanged and devices with rediscovery disabled are not affected.

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

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 to 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 allows **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 only 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", [...]
```

deviceDependency.x

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 the wildcard for the `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>" &#124; "deviceSerial=<value>"||]
```

dge.x

Also see the commands for DGE-extensions below (dgex.create, etc)

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.delete

Deletes configuration information for one or more existing DGE instances.

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

dge.list

Lists DGE information based on search criteria.

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

dge.update

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

```
dge.update <"dgeName=<regexp>" | "dgeSerial=<value>">
```

```
[, "host=<new_value>"]
[, "locationName=<new_value>"]
[, "softLimit=<new_value>"]
[, "hardLimit=<new_value>"]
```

dgeX.x

These commands are for DGE-extensions. Note that there is no `Location` parameter for DGE-extensions. These commands are only available when logged into the API as superuser.

dgeX.create

Creates a new DGE extension.

The `dgex.create` command is available only when logged into the BVE API server as `superuser`.

```
dgeX.create "dgexName=<value>"
, "description=<value>"
, "softLimit=<value>"
, "hardLimit=<value>"
[, "upstreamDgeName=<value>"]
```

The value for `upstreamDgeAddress` defaults to the configured host IP address of the upstream DGE. If the upstream DGE has multiple IP addresses, make sure to set `upstreamDgeAddress` to the IP address that is reachable by the DGE-extension.

Example

```
dgeX.create "dgexname="Cust-12" , "description=Acme Company HQ",
"softlimit=200", "hardlimit=500", "upstreamDgeName=Central-DGE-2",
"upstreamDgeAddress=192.168.10.222"
```

dgeX.delete

Deletes a DGE extension.

The `dgex.delete` command is available only when logged into the BVE API server as `superuser`.

```
dgeX.delete "dgexName=<value>" | "dgexSerial=<value>"
```

dgeX.update

Updates an existing DGE extension.

The `dgex.update` command is available only when logged into the BVE API server as `superuser`.

```
dgeX.update <"dgexName=<value>" | "dgexSerial=<value>">
```

If both `dgexName` and `dgexSerial` are specified, only `dgexName` is updated.

dgeX.list

```
dgeX.list <"dgexName=<value>" | "dgexSerial=<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 the `eventType` parameter, it is possible to display only events where the previous or current state was of that type.

```
event.list
["deviceName=<regexp>"]
[, "testName=<regexp>" | "testSerial=<value>"]
[, "startTime=<YYYYMMDDhhmm>"]
[, "endTime=<YYYYMMDDhhmm>"]
[, "eventType=<ok|warning|critical|fail|unreachable|unknown>"]
[, "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 | event_duration | previous_state |
new_state | event_message | last_test_result
```

The `event_duration` is provided in milliseconds.

location.x

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.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>"]
```

location.update

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

```
location.update
[, "streetAddress=<new_value>"]
[, "city=<new_value>"]
[, "state=<new_value>"]
[, "comments=<new_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 |
```

sla.x

sla.create

Create a new SLA for a container, device or test.

```
sla.create slaName=name, slaType=<container|device|tests>,
calculationPeriod=<day|week|month>,
containerName=<container> | deviceName=<device> | testIDs=<tid1;tid2;tid3>,
threshhold=<percent>
[, startTime=YYYYMMDD[hhmm] ]
[, minGranularity=<minute|hour|day|week> ]
```

```
[, comment=<string> ]
[, scheduleName=<schedule name> ]
[, permitPast=<true|false> ]
```

The `minGranularity` parameter is used to limit the drill down into statistics from the front end to the specified level. If you want **Traverse** to calculate the SLA for historical data, you can specify a `startTime` in the past, and set the `permitPast=true`.

For example, assume a container called Email Container has already been created using the web interface. To create an SLA calculated monthly for a schedule from 9 to 5, Monday through Friday:

```
sla.create "slaName=email Service SLA", "slaType=container",
"calculationPeriod=month", "containerName=Email Container",
"startTime=201105150000", "threshold=98.99", "comment=SLA for Exchange
Service", "scheduleName=Business Hours"
```

sla.update

Update an existing SLA.

```
sla.update <slaName=name|slaSerial=serial>
```

```
[,newName=<string>]
[,comment=<string>]
[,threshold=<percent>]
[,minGranularity=<week|day|hour|minute>]
[,scheduleName=<value>]
```

Note that you cannot change the `calculationPeriod` of an existing SLA test.

sla.delete

```
sla.delete <slaName=name|slaSerial=serial>
```

sla.list

```
sla.list <slaName=name|slaSerial=serial>
```

sla.status

```
sla.status <slaName=name|slaSerial=serial>
[,calculationPeriod=<day|week|month>]
[,startTime=YYYYMMDD[hhmm] ]
[,endTime=YYYYMMDD[hhmm] ]
[,maxResults=n]
```

test.x

test.create

Creates new tests for an existing device.

```
test.create "deviceName=<new_name>"
, "testType=<ping | snmp | port | external | composite | ...>"
, "subType=<based on testType>"
, "testName=<test name displayed on screen>"
, "warningThreshold=<value>", "criticalThreshold=<value>"
, "units=<string>"
, "resultMultiplier=<number>"
, "resultProcessDirective=<see table below>"
, "maxValue=<number>"
[, "actionName=<new_value>"]
```

Some of the parameters are specific to each test type. For a detailed list of commands and different test types, type `help test.create`.

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.

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.
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).
8	TIMETICKS.	Convert number of milliseconds since midnight Jan 1, 1970 into dd hh:mm:ss format.

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. For 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>"]
```

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.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=<regexp>"
```

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

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.list

Displays test configuration parameters for tests matching search criteria.

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

Sample output. The 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","max
Value=101089",
"snmpOid=.1.3.6.1.2.1.25.2.3.1.6.2"
```

To get test names for a device, use the following command:

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

test.resume

Resumes regular testing for one or more previously suspended tests.

```
test.resume <"deviceName=<regexp>" | "deviceSerial=<value>">
, <"testName=<regexp>" | "testSerial=<value>">
[, "testType=<regexp>"]
[, "subType=<regexp>"]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=<regexp>" | "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.

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—for example, from Warning to Critical or from Unknown to Unreachable—the suppression is reset automatically.

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

test.suspend

Suspends testing of one or more existing tests.

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

test.update

Updates configuration information for one or more existing tests.

```
test.update <"testName=<regexp>"
, "deviceName=<regexp>"
, "testType=<value>"
[, "subType=<regexp>"]
[, "interval=<new_value>"]
[, "warningThreshold=<new_value>"]
[, "criticalThreshold=<new_value>"]
[, "actionName=<new_value>"]
[, "maxValue=<new_value>"]
[, "units=<new_value>"]
[, "testSerial=<value>"]
[, "resultProcessDirective=<value>"]
[, "thresholdType=<auto|ascend|descend|discrete|-1|1|2|3>"]
[, "resultMultiplier=<value>"]
[, "userName=<value>"]
[, "resultProcessDirective=<value>"]
```

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

```
test.update "devicename=*router*", "testname=Round Trip*", "testtype=ping",
"subtype=rtt", "warningthreshold=150", "criticalthreshold=250"
To modify a test name for a device named "abc-switch", first use the test.list
command to obtain the serial number for the test, and then use the following
command:test.update "devicename=abc-switch", testname="New Test Name",
"testSerial=123456", "testtype=ping"
```

user.x

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.delete

Deletes a user/login id from a specific department.

```
user.delete <"loginName=<regexp>" | "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>"
```

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=<regexp>" | "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>"]
```

userClass.x

userClass.create

Creates a user group.

```
userClass.create "groupName=<new_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>"]
```

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>"]
```

Further Examples

Creating a New Device and Test

```
% telnet bve_host 7661
```

```
Trying 127.0.0.1...
```

```
Connected to localhost.
```

```
Escape character is '^]'.
```

```
OK 200 Traverse BVE TCP Server v5.0 ready
```

```
LOGIN zyrion/zyrion
```

```
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"
```

Creating a New Test Container and Placing It in a New Device Container

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

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

```
container.create "serviceName=All Devices", "serviceType=device",
"memberListMethod=manual", "memberList=#All RTT Tests"
```

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

```
container.members "serviceName=All Devices"
```

```
OK 203 request accepted, records returned: 1
```

```
"serviceName=All Devices", "serialNumber=300003", "memberType=container",
"memberName=All RTT Tests", "memberStatus=Unreachable", "deviceName=All RTT
Tests", "deviceSerialNumber=300000", "accountName=My_Company",
"accountSerialNumber=49"
```

C h a p t e r   2

Traverse REST API

In This Chapter

REST API Overview

The **Traverse** Representational State Transfer (REST) provides another way to access all the functionality available in the BVE FlexAPI. All the commands and responses that are present in the FlexAPI are available using the REST protocol as well.

What is REST ?

REST (REpresentational State Transfer) is an approach for building services that make specific resources available at a URL. A REST service has well defined operations for manipulating the resource. Typically, these operations include reading, writing, editing, and removing.

REST Command Format

To access the REST API, connect to the **Traverse** Web Application URL with the additional path as described below:

REST Command Format

```
<protocol>://<host>:<port>/api/rest/command/<command>?<params>
```

where:

<protocol>	Either http or https, depending on how your Traverse webapp is configured.
<host>	Hostname or IP address for your Traverse webapp.
<port>	Normally not required. Use when running Traverse on a non-standard path.
<command>	The REST API commands.
<params>	Any parameters required by the <command>.

For example:

```
http://traverse.my.domain/api/rest/command/login?john/mypassword
```

To access the WADL:

```
http://<host>:<port>/api/rest?_wadl&_type=xml
https://<host>:<port>/api/rest?_wadl&_type=xml
```

REST Commands in Traverse

All BVE commands are supported. All responses are exactly similar in format and data to the BVE FlexAPI.

Login

In BVE

```
login user/pass
```

In REST

```
http://host:port/api/rest/command/login?user/pass
```

REST Response

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

WhoAmI

In BVE

```
whoami
```

In REST

```
http://host:port/api/rest/command/whoami
```

REST Response

```
OK 203 request accepted, records returned: 1
"serialNumber=1870003", "loginName=john", "departmentName=Zyrion, Inc.",
"role=Read-Write", "lastName=smith", "firstName=john", "emailAddress=",
"timeZone=GMT", "locale="
```

Logout

In BVE

```
logout
```

In REST

```
http://host:port/api/rest/command/logout
```

REST Response

```
OK 299 Logging out.
```

Devices.List

In BVE

```
devices.list "deviceName=*U*", "address=192.168.10.*"
```

In REST

```
http://host:port/api/rest/command/devices.list?deviceName=*U*&address=192.
168.10.*
```

REST response

```
OK 203 request accepted, records returned: 9
"serialNumber=280261", "deviceName=Primary Distribution Switch",
"address=192.168.10.251",
"deviceType=Network Switch", "parentNames=Cisco Router",
"snmpCid=in72days36move", "snmpPort=161",
"snmpVersion=2", "snmpOptimize=1", "agentBatchMode=1", "agentPort=161".....
```

Chapter 3

External Data Feed (EDF) Reference

In This Chapter

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 be used to access the EDF instead of a direct telnet connection for consistency.

Connecting To The Server

Communication with the EDF server consists of two phases: a connection establishment phase and a command-execution phase. During the connection establishment phase the 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 are 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 from the server, possibly consisting of multiple lines of output.

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. The 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** *(page 36)*. 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 is printed in the form `ERR reason` and the server disconnects the client.

Disconnecting From the Server

When the client application disconnects from the EDF Server, it is recommended that the client issue a disconnect request instead of simply closing the socket connection. This will allow the server to perform proper cleanup before disconnecting the session. See **Client Commands** *(page 37)*.

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`.

Command/Reply Formatting and Commands

The commands sent by a client and the 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 ... ]`.
- A pipe symbol (`|`) is not permitted as part of the value.
- For each client command, the server responds with a response code indicating success or failure, and optionally includes some descriptive text indicating the 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 in the following format:
`<status code>` optional informative text
where status code is one of the following:

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

Client Commands

Login

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

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

Logout | Quit

Ends a login session.

```
session.Logout
```

Result.insert

Inserts a result value for an existing test into the database. The `Result.insert` must be submitted to the EDF listener on the DGE that is monitoring the device.

```
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 `nnnnnnnnnn` format where `nnnnnnnnnn` is the 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 is 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.result_value` may be set to several special values to represent differing states in **Traverse**:

-1	UNKNOWN	Indicates an unexpected, or unknown test result
-2	FAILED	Used when a test fails, such as an http test returning a 500 code
-3	UNREACHABLE	
-4	SUSPENDED	Test is not being run

Example

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 v5.0 ready
```

```
LOGIN traverse/traverse
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
```

Now connect to the EDF server on port 7657 on the DGE that is monitoring the device. Use 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
```

To view the newly inserted test result via BVE Server:

```
% telnet bve_host 7661
OK 200 Traverse BVE TCP Server v5.0 ready
```

```
login zyrion/zyrion
```

```
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|20030506100124|1|25|25|2
5|Ok|55|85
```

```
QUIT
```

```
OK 299 Logging out.
```

Templates for EDF Tests

You can set up templates for EDF tests. If you create an XML configuration file under the `plugin/monitors` directory `my_edf_test.xml` and restart the web application and DGE components, you will see the defined tests in the Administration > Devices > Tests > Create New Advanced Tests > External Data Feed (API) 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`.

EDF versus 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.

Examples

Powershell Script

Communicate to Cisco Router using EDF

```
Param(

[parameter(position=0,Mandatory=$true)][validatenotnull()][String]$devicen
ame,

[parameter(position=1,Mandatory=$true)][validatenotnull()][String]$devicei
p,

[parameter(position=2,Mandatory=$true)][validatenotnull()][String]$testnam
e,

[parameter(position=3,Mandatory=$true)][validatenotnull()][String]$edfcoun
ter
)
function readResponse {
  while($stream.DataAvailable)  {
    $dataLength = $stream.Read($buffer, 0, 1024)
    Write-Host $encoding.GetString($buffer, 0, $dataLength)
  }
}
function insert-result
{
  Begin
  {
    ## Open the socket, and connect to the computer on the specified port
    $socket = new-object System.Net.Sockets.TcpClient("192.168.10.21", 7657)
    if($socket -eq $null) { return; }
    $socket.SendTimeout = 10
    $socket.ReceiveTimeout = 10
    $stream = $socket.GetStream()
    $writer = new-object System.IO.StreamWriter $stream
    ## Log into the EDF server
    $writer.WriteLine("login edfuser | fixme")
    $writer.Flush()
    Start-Sleep -m 1000
    readResponse($stream)
    ## Insert test result
    $writer.WriteLine("result.insert $devicename | $deviceIP | $testName | |
0 | $edfCounter")
    $writer.Flush()
    Start-Sleep -m 1000
    readResponse($stream)
    ## Log out of the server
    $writer.WriteLine("logout")
    $writer.Flush()
    ## Close the streams
    $writer.Close()
    $stream.Close()
  }
}
$buffer = new-object System.Byte[] 1024
$encoding = new-object System.Text.AsciiEncoding
. insert-result
```

Chapter 4

Traverse Perl API

In This Chapter

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.

Zyrion::ExternalData - EDF API

This Perl module provides a programmatic interface into the monitoring framework of **Traverse** using the External Data Feed (EDF) API. It can be used to connect to a remote server (DGE), create an authenticated session, and insert test results for existing (previously provisioned) tests.

new

Create a new `Zyrion::[ExternalData]` object.

```
use Zyrion qw([ExternalData]);
my $obj;
my $host_name_or_ip = "192.168.10.131";
my $tcp_port = "7657";
my $login_id = "edfuser";
my $login_password = "fixme";
my $debug = 1;
$obj = Zyrion::[ExternalData]\->new(
   Host => $host_name_or_ip,
   Port => $tcp_port,
   DEBUG => $debug);
```

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 the `Login` method. The object is created with remote host address and port information, but no connection is made to the remote host when this method is called. 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 in to the **Traverse** EDF server.

```
my $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 are

different from those 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=>"$device_fqdn_or_ip",
dateTime=>time,
testName=>'[ExtTest]',
result=>$value_to_insert,
extraInfo=>$additional_info);
Refer to&nbsp;[External Data Feed (EDF) Reference on page
51|file://share/Documents/Documentation/FrameMaker-Docs/TechPubs55/Dev%20G
uide/HTML/Traverse%20Developers%20Guide-4-1.html#wp1083882]&nbsp;for
explanations of the parameters and valid values.
```

Example: Connect, Log In, Insert Test Result, Log Out

The following example creates a connection to `localhost` (default port), logs in, inserts a result for a test named `sample_test` on the device with IP address `192.168.200.50` and name `my_server` into the DGE database, and logs out.

Note: *You should use a test name, device address, and device name appropriate to your installation. You may also need to change the* `Login` *user and password if they have been changed from the defaults.*

```
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,
extraInfo=>'needs immediate action') \|\|
print "ERROR: ", $obj->[GetErrorMsg], "\n";
$obj->Logout;
```

Note that the optional parameter `testSerial` was not provided, so the server will use `deviceName`, `deviceAddr` and `testName` to uniquely identify the test.

Zyrion::Message - ISM API

This Perl module provides a programmatic interface into the messaging framework of **Traverse** using the Input Stream Monitor (ISM) API. It can be used to connect to a remote server (DGE), create an authenticated session, and insert fixed format or free-form messages (events) against existing (previously provisioned) devices.

new

Create a new `Zyrion::Message` object.

```
use Zyrion qw(Message);
$obj = new Zyrion::Message(
  [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. The object is created with remote host address and port information, but no connection is made to the remote host when this method is called. 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

Logs in to the **Traverse** ISM server.

```
$return_value = $obj->Login( );
```

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 are different from those 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

Logs 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](
[severity   => <"ok"|"warning"|"critical">,]
);
```

or,

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

The first form of the method inserts a message specific for the specified device into the system for further processing. The second method forces the system to match the message against all configured regular expression patterns (ruleset). If there is a match, appropriate severity is set and actions are 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.

Example: Connect, Log In, Insert Message, Log Out

The following example creates a connection to `localhost` (default port), logs in, inserts a message for the device with IP address `192.168.200.50` into the DGE database, and logs 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=>'warning',
Message=>"this is a test") \|\|
print "ERROR: ", $obj->[GetErrorMsg], "\n";
$obj->Logout;
```

Zyrion::Provisioning - BVE API

This Perl module provides a programmatic interface into configuration and historical performance data of **Traverse** using the Business Visibility Engine (BVE) API. It can be used to connect to a remote server (BVE), create an authenticated session, and perform create/delete/update tasks on various **Traverse** objects (user, device, test, etc.), as well as get real-time test details and reports.

The detailed list of commands and parameters expected by the BVE socket server is detailed in BVE FlexAPI Protocol Reference.

new

Create a new `Zyrion::Provisioning` object.

```
$obj = new Zyrion::Provisioning(
\[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 the `Login` method. The object is created with remote host address and port information, but no connection is made to the remote host when this method is called. 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`.

CreateX, ListX, UpdateX, DeleteX, SuspendX, ResumeX, ExportX, MoveX

```
$return_value = $obj->CreateX (
```

These methods allow manipulation of different **Traverse** objects (X). Valid objects include the following:

- Action
- `[AdminClass]`
- Container
- Department
- Dependency
- Device
- DGE
- Location
- Test
- User
- `[UserClass]`

The parameters for each object and method combination are different. Refer to **BVE FlexAPI Protocol Reference** *(page 4)* 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 the `GetErrorMsg` method. On success, a non-zero return value is provided. Results for `ListX` methods are stored in an internal array and accessed using the `GetResultRef` method.

GetContainerMembers

```
$return_value = $obj->[GetContainerMembers](
```

This method returns a list of objects inside one or more containers. For a device container, the list may contain devices and/or other (nested) containers. For a test container, the list will contain tests and the device/department of those tests. The `GetResultRef` method should be used to access the data. Since a single container will most likely have multiple members, the parameters will be returned in an array, as in the following example:

```
$result_ref = $obj->[GetResultRef]();
foreach $serial_num (keys %{ $result_ref }) {
  $num_members = scalar(@{ $result_ref->{$serial_num}->{membername} })
  foreach $this_member (0 .. ($num_members - 1)) {
    foreach $object_param (keys %{ $result_ref->{$serial_num} }) {
      $param_value = $result_ref->{$serial_num}->{$object_param}[];
    }
  }
}
```

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`, `ExportX`, `MoveX`). 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

Returns a pointer to the 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 the result of an earlier `List<object>` method (see `CreateX`, `ListX`, `UpdateX`, `DeleteX`, `SuspendX`, `ResumeX`, `ExportX`, `MoveX`). Each `List<object>` method stores results in the same internal buffer, so you should store or 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 following format:

```
$result_ref->{<serial_number>}\->{dgename} = "dge01.eng"
\->{locationname} = "Default Location"
\->{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 the 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 the result of an earlier `List<object>` method that returned results in bulk format ("`|`"-separated list). Each `List<object>` method stores results in the same internal buffer, so you should store or process the results of one search before executing a new search.

GetXStatus

```
$return_value = $obj->GetXStatus(
```

These methods allow retrieval of current overall health for monitored objects (X). Valid objects include the following:

- Container
- Device
- Test

Login

Logs in to the BVE API server.

```
$return_value = $obj->Login();
```

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 to a different user in the BVE API server and assume the new user's permissions and privileges.

Logout

Log out of the BVE API server.

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

This method sends a logout command to the BVE API 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.

Examples: Connect, Log In, Create a DGE, Log Out

This example creates a connection to `localhost`, logs in, creates a new DGE, and logs out.

```
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";
$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, the 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:

```
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 = ();
$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) ¦¦
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.

GetErrorMsg

Retrieve error information from last operation.

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

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

Further Examples

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

Chapter 5

Plugin Monitors

In This Chapter

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.

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 is ignored and an error message logged. Such configuration information is loaded from `$TRAVERSE_HOME/etc/TestTypes.xml` and other plugin configuration files, which are 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 `$TRAVERSE_HOME/etc/TestTypes.xml` or other directories should not be edited (unless you are instructed to edit them by **Kaseya Support** *(https://helpdesk.kaseya.com/home)* as it may adversely affect or cause failure of **Traverse** components. Any changes made to these directories may also be lost when a new version of **Traverse** is installed. All user customizations are expected to placed in `$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:

XML Testtype Child Elements

Child Element	Description
displayName	A user-friendly name that is used when creating a report or referring to a specific testtype
displayCategory	This setting defines the column that the test result should be in on the summary pages in the Web application. Valid values are network, system, and application.
subType	This is a string that uniquely identifies the testtype 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.
units	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.
severityAscendsWithValue	This is used to indicate a severity direction for test values, and has the following possible values: true false or static If the value is true, then the status being tested becomes more critical as the test value rises. When the

	value is static, you can set discrete threshold values for warning and critical.
defaultWarningThreshold	This is the default end user warning threshold for this test type. If the severityAscendsWithValue is `static', then you can specify a comma separated set of numbers using the following syntax:1,3,5,8-20
defaultCriticalThreshold	This is the default end user critical threshold for this test type.
showAsGroup	Group tests of the same sub-type together in the Web application during autoDiscovery of SNMP tests for a device.
shadowWarningThreshold	The default *admin* warning threshold for this test type. Typically this value will be same as defaultWarningThreshold.
shadowCriticalThreshold	The default *admin* critical threshold for this test type. Typically this value will be same as defaultCriticalThreshold.
slaThreshold	The default SLA threshold for this test type.
testInterval	The default interval, in seconds, for running this test.
testField	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:fieldName - This will be used as key for the field value when it's passed to the test.fieldDisplayName - A user friendly name for the field that will be used by the Web application when creating or updating tests.isRequired - This element indicates whether or not the a value is required to be given for the field when creating or updating the test.isPassword - 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.defaultValue - A default value that will be presented to a user when creating the test.

Post Processing for Plugin Monitors

Unlike built-in monitors, post-processing directives are not allowed for plugin monitors. The monitors are supposed to do all the processing and return the final result.

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 format 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:

Monitor Element Attributes

Attributes	**Description**
type	This defines a type name for the plugin, and the type of monitoring it does. The value of type will show up in the DGE status line when displaying the testing queues and monitor status, and will be used in the Web application
plugintype	This attribute describes the type of plugin. For a Java plugin monitor, this parameter should be set to java.
resource	This is the name of the resource of that should be used to do the tests. For a plugintype of java, this should be the fully qualified name of a Java class file that implements the [NetvigilPlugin] 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 creates a new instance of your `[NetvigilSimplePlugin]` subclass and calls 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` is stored in the DGE database, and used for reports and status. If the value returned is `RESULT_UNKNOWN` or `RESULT_FAILED`, the DGE calls `getErrorMessage` and puts 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 creates one instance of the plugin when it starts up, and calls `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 calls `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 calls the `runBatch` method of the plugin object. When `runBatch` returns, the DGE calls `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 takes the results and stores them in the DGE database, where they are used in reports and status displays. If any of the results has a value of `RESULT_UNKNOWN` or `RESULT_FAILED`, the DGE calls `getErrorMessage` with the index of the result in the array returned by `getTestResults`. If the error message returned is not empty, it is 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* (page 56), 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 must be restarted before the new monitor is usable. When **Traverse** starts, will scans the monitor plugin directory for XML and `.jar` files. It adds the tests described by the XML file to its list of test descriptions, and adds the `.jar` files found in `<type>/lib` to the Java `CLASSPATH`. If an XML file has an error in it, a message is written to the error log, and the XML file is ignored. Each different XML file results in a separate test queue in the **Traverse** DGE.

Creating a Plugin Test

1. Navigate to Administration > **Devices**.
2. Click **Tests** on the line for the device you want to add tests for.
3. Click Create **New Standard Tests**.
4. Select the radio button for **Create new tests by selecting specific monitors.** You should now see the tests you defined in the plugin XML file.
5. Check the check box for each plugin monitor test you want to provision.
6. Click **Add Tests**.

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.

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:

Monitor Element Attributes

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

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 associates 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 `}`.

The following table shows the variables that can be used in the `parameters` element of the configuration file.

Variables Available for the parameters Element

${container_name}	${container_member_count_all}
${container_member_count_match}	${affected_containers}
${department_name}	${product_name}
${device_comment}	${device_address}
${device_location}	${device_model}
${device_name}	${device_serial_number}
${device_snmp_cid}	${device_snmp_version}
${device_tag1}	${device_tag2}
${device_tag3}	${device_tag4}
${device_tag5}	${device_tag1_caption}
${device_tag2_caption}	${device_tag3_caption}
${device_tag4_caption}	${device_tag5_caption}
${device_type}	${device_vendor}
${test_name}	${test_serial_number}
${test_shadow_critical_threshold}	${test_shadow_warning_threshold}
${test_sla_threshold}	${test_sub_type}
${test_type}	${test_units}
${test_user_critical_threshold}	${test_user_warning_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.

Note: `\${device_tag1}` provides the value configured for `tag1` for the device. `\${device_tag1_caption}` provides the description (caption) of `tag1` as configured in the `emerald.xml` file.

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 output 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 (UNKNOWN), or
- `-2` to indicate that the test failed for a known reason (FAIL).

Note: The plugin monitor takes the last numerical value on a line by itself as the test result.

You can also pass out debugging or error information from your script. Any lines beginning with the string `DEBUG:` is logged to the `monitor.log` file under `$TRAVERSE_HOME/logs/` directory. Any lines beginning with the string `ERROR:` are logged to the `error.log` file in the `logs` directory with `WARN` severity.

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 must 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.

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_with_value>
<defaultWarningThreshold>2,5</defaultWarningThreshold>
<defaultCriticalThreshold>4,8-10,99</defaultCriticalThreshold>
<shadowWarningThreshold>2,5</shadowWarningThreshold>
<shadowCriticalThreshold>4,8-10,99</shadowCriticalThreshold>
<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 is in a 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:

a. Log in as end user.
b. Click on Administration > Devices > **Tests**.
c. Click on **Add New Standard Tests**.
d. Check **Atmosphere**.
e. Enable the test, and make sure that **discrete** is selected as a severity option.
f. Submit the form.

4. 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
```

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` where `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** component after editing or creating new files.

C h a p t e r 6

External Authentication

In This Chapter

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 the **Traverse** provisioning database. So it will be necessary to create the login ID on the **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`. Otherwise, older users will continue to be authenticated by the old mechanism (e.g., the **Traverse** internal password database).

> **Note:** Changes to `emerald.xml` need to be reinstated on an upgrade since all changes might not be copied over during an upgrade.

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` is 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** *(page 65)*.

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.

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 is called. The authentication string this method returns is stored in the provisioning database, along with the unique string you picked to identify the authentication method. **Traverse** uses 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 is retrieved from the database, and it, along with the password the user gave and the user login name is passed to your `[NetVigilPluginAuthentication].authenticate` method. If your authenticate method returns true, the user is allowed to login. Conversely, if authenticate returns false, the user is not able to login.

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

Authentication Plugin Script

You can also specify a script, program or batch file to use for authentication. When **Traverse** runs the program the user's login name and password are passed 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 of the arguments passed to your program.

Here's an example Perl script (`auth.pl`) that only lets 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 (`$TRAVERSE_HOME/plugin/auth`). To instruct **Traverse** to use your script for authentication, you'll need to modify `$TRAVERSE_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 tells **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* (page 64). 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 `${username}` and `${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 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 on the command line may be viewed by anyone on your system with the `ps` command during the time it takes the script to execute.*

Samples for Windows AD, Radius

Sample scripts for authentication using Windows Active Directory, Radius, etc. have been contributed by **Traverse** users and are available on the Kaseya Community web site by searching for the keywords `authenticate against active directory`.

Web URL 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/utils/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—for example, to make one report publicly available—without allowing a full login.

Chapter 7

Plugin Actions

In This Chapter

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.

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.

writeToFile.xml

Here is a sample configuration file.

```
<?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}
* ${action_profile}
* ${action_class}
* ${device_tag1}
* ${device_tag1_caption}
* ${timestamp}
* ${device_name}
* ${device_address}
* ${device_vendor}
* ${device_snmp_cid}
* ${device_location}
* ${current_shadow_severity}
* ${time_in_state}
* ${container_member_type}
* ${container_member_count_match}
* ${container_name}
* ${affected_containers}
* ${container_member_summary}
```

- `$container_member_count_match` gives the number of immediate children in the container having the same severity as the container.
- `$affected_containers` gives the parent containers impacted by the severity of the current containers (one per line).
- `$container_member_summary` includes details on the severity of the container's children.
- `${device_tag1}` provides the value configured for `tag1` for the device.
- `${device_tag1_caption}` provides the description (caption) of `tag1` as configured in the `emerald.xml` file.

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

```
- ${test_name}
- ${test_user_warning_threshold}
- ${test_sla_threshold}
- ${test_sub_type}
- ${result_value}
* ${test_serial_number}
* ${test_user_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`.

writeToFile.sh

Here is a sample script that corresponds to the sample configuration file provided above.

```
#!/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 must be restarted. Once the configuration file has been loaded, it show ups on the drop-down list in the 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, using different parameters. To do this, create multiple plugin action configurations that correspond to the same script.

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 must 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 in to 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 in to and rebooting a Cisco router.

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 confided 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 a certain subject created using a 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 **Kaseya Support** *(https://helpdesk.kaseya.com/home)* 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 DGEs, this will assist you in compiling a list of DGEs where the package needs 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. Iif the web application is running on a separate host, there is no need to install RT on that host. Install RT under it's 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 DGEs. 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 ensures that when a new ticket is created, no auto-replies are sent, if there is such a script configured for the queue you will be using.
- Ensure 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 must 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.

- Ensure 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

To install the *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 copies the integration package into the appropriate location under **Traverse** installation directory.
4. You must restart the DGE process and web application at a convenient time before the action becomes visible in the drop down list in the web application, or can be executed by a DGE.

Configuration on Windows

To use the newly added plugin action, you must first 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.

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

Start > Programs > Traverse > **Stop Traverse**. 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.

By default, a new ticket is 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 is entered into the same ticket and the second test information is added as additional comment, then edit the XML configuration file for the script and add the `--perdevice` option to the `<parameters>..</parameters>` section.

Configuration on UNIX

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.

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

```
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.

By default, a new ticket is 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 is entered into same ticket and the second test information is added as additional comment, then edit the XML configuration file for the script and add the `--perdevice` option to the `<parameters>..</parameters>` section.

Troubleshooting

Look in the `traverse/logs/error.log` for any error messages logged by the DGE process.

(UNIX) Check to make sure `plugin/actions/createTicketInRT.pl` is executable (mode 0555).

(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
```

Chapter 8

Web Services API

In This Chapter

Overview

his chapter provides an overview of the **Traverse** Web Services Application Programming Interface (API). You can use the **Traverse** Web Services API to create your own portal to provide a customized view of the system. For example, you could create a portal showing a limited number of devices or an aggregate status showing the health of the network.

The **Traverse** Web Services API consists of the following 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.
Department User Container Device Test DGE (includes location) Event Subnet Search Types	These web services allow read-only access to the different data within Traverse for use in your applications. For a complete listing, see the **javadoc** *(http://www.zyrion.com/support/docs/v5.5/javadoc/)*.
External Data Feed	This web service allows inserting tests data into Traverse.

Important: You should also view the WSDL for the most current API definitions.

Traverse Web Services API Workflow

The following describes a typical **Traverse** Web Services API workflow:

1. Client application uses the Session Manager service to log in to **Traverse** by providing a username and a password.
2. **Traverse** sends back a response, including a session identifier (ID). This session identifier can also be shared with the web application.
3. The client application uses the session ID to make calls to **Traverse** through the different web services. 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.

Web Services Operations

Depending on login credentials, **Traverse** exposes a certain set of objects. Each web service operation has the same basic structure:

- They have a single request parameter and they return a response parameter.
- The request objects all have a mandatory session ID along with the rest of the arguments.
- The response objects all have an integer `statusCode` and `statusMessage` field. The `statusCode` should always be checked when receiving a response. A value of `0` indicates success, and any other value is either a warning or error with the description in the `statusMessage` field.

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.
- `Admin user` - An admin user has limited administrative access to the system.

- `Superuser` - A superuser has full access to the system.

Objects in Traverse

The following types of objects are available in **Traverse**:

- Departments
- Containers
- Network Devices
- Test Configs

Each object is uniquely identified by a serial number, which is a positive 64-bit integer value. While object names can change, the serial number should be used as a key to uniquely identify objects.

Object Severity & Status

The status of the different objects is represented by an integer value:

Status	Integer Value	Description
UNCONFIGURED	0x00000008	Device is provisioned, but no tests have been created.
SUSPENDED	0x00000080	Test is not being run.
OK	0x00000800	Object is in OK state.
TRANSIENT	0x00004000	Object is flapping between OK and non-OK state.
UNKNOWN	0x00000800	
UNREACHABLE	0x00080000	
WARNING	0x00800000	Object is in Warning (yellow) state.
CRITICAL	0x08000000	Object is in Critical (red) state.
FAIL	0x40000000	

Time Expressions

There are two ways to specify the time using **Traverse** Web Services.

- The first way is to enter a positive 64 bit integer representing the number of milliseconds since 1/1/1970, or Epoch time times 1000. This is typically used in the `startTime` and `endTime` parameters of applicable requests.
- The second way is to use a relative date. The time expressions contain 3 pieces of information separated by a dash (`-`). This is typically used in the `startTimeExp` and `endTimeExp` parameters of applicable requests.

The format is:

```
number-unit-direction
```

where:

- `number` = any positive integer
- `unit` = minutes, hours, days, weeks, months, years
- `direction` = `ago` or `fromnow`

For example, to run a report for the last 24 hours:

```
startTimeExp = 24-hours-ago endTimeExp = now
```

Object Filter

As its name implies the object filter is used as a part of the request parameter to filter the results. The object filter is based on the search workflow found throughout the application. A very large number of fields can be used to search tests, network devices and hierarchy containers as listed in the table below. For a complete listing, see the **javadoc** *(http://www.zyrion.com/support/docs/v5.5/javadoc/)*.

Name	Type	Description
departmentName	String	The name of a department
departmentSerialNumber	List<Long>	A list of department serial numbers.
deviceName	String	Name of a network device
deviceAddress	String	The IP address of a network device
deviceTypes	List<Integer>	A list of device types. Device types can be one of the following:0. NT/Windows1. Unix/Linux2. Switch3. Router4. Firewall5. SLB6. Proxy7. VPNC8. Printer9. Wireless10. Unknown11. Storage12. VMWare
deviceSerialNumbers	List<Long>	A list of network device serial numbers
testTypes	List<String>	Type of monitor running the test. One of:ping snmp wmi port radius ntp dns sql sql_value ldap external deepweb jmx flow oracle pgsql cmr vmware
testSubTypes	List<String>	

Each testType has a series of sub-types. Most common ones are:

- `rtt` - Ping round trip time
- `pl` - Ping packet loss
- `cpu` - WMI or SNMP CPU load
- `phymemory` - Physical memory usage

`bandwidth` - Link utilization (%) For a complete listing, see the **javadoc** *(http://www.zyrion.com/support/docs/v5.5/javadoc/)*.

testSerialNumbers	List<Long>	A list of test serial numbers
testStatuses	List<Integer>	A list of severities in which a test currently is.
deviceStatuses	List<Integer>	A list of severties in which a device currently is.
containerName	String	The name of a container that a device or test is a part of
elementName	String	The name of an element that the test is a part of or contained inside a device.
elementSerialNumbers	List<Long>	A list of serial numbers of elements that tests are part of or contained inside a device.
elementCategory	String	The category of an element that tests are a part of or contained inside a device.
excludeExportedDevices	Boolean	Whether or not exported devices and their tests should be excluded from the search filter.

[ObjectFilter] Class

Traverse WSDL Files

The different Web Service WSDL files are available at the following locations (replace localhost with the hostname running the **Traverse** BVE):

Service	Location	
Container service:	http://localhost/api/public/container?wsdl	http://localhost/api/public/container?wsdl=ContainerService.wsdl
Department service:	http://localhost/api/public/department?wsdl	http://localhost/api/public/department?wsdl=DepartmentService.wsdl
Device service:	http://localhost/api/public/device?wsdl	http://localhost/api/public/device?wsdl=DeviceService.wsdl
DGE service:	http://localhost/api/public/dge?wsdl	http://localhost/api/public/dge?wsdl=DgeService.wsdl
Event service:	http://localhost/api/public/event?wsdl	http://localhost/api/public/event?wsdl=EventService.wsdl
External data feed:	http://localhost/api/public/edf?wsdl	http://localhost/api/public/edf?wsdl=ExternalDataFeedService.wsdl
Search service:	http://localhost/api/public/search?wsdl	http://localhost/api/public/search?wsdl=SearchService.wsdl
Session manager (new location):	http://localhost/api/public/sessionManager?wsdl	http://localhost/api/public/sessionManager?wsdl=SessionManagerService.wsdl
Subnet service:	http://localhost/api/public/subnet?wsdl	http://localhost/api/public/subnet?wsdl=SubnetService.wsdl
Test service:	http://localhost/api/public/test?wsdl	http://localhost/api/public/test?wsdl=TestService.wsdl
Types service:	http://localhost/api/public/types?wsdl	http://localhost/api/public/types?wsdl=TypesService.wsdl
User service:	http://localhost/api/public/user?wsdl	http://localhost/api/public/user?wsdl=UserService.wsdl

Sample Code

Sample code can be found online in the **Traverse community** *(http://community.zyrion.com/)* web site. Some examples for using the Web Service calls are described below.

Session Establishment

```
// Services
SessionManagerService loginService;
TypesService          typesService;
DeviceService         deviceService;
ContainerService      containerService;
TestService           testService;
UserService           userService;

//Login
LoginRequest request = new LoginRequest();
request.setUsername( "zyrion" );
request.setPassword( "zyrion" );
request.setLoginType( LoginType.USERNAME );
request.setSessionType( SessionType.Traverse );
request.setRemoteAddress( "127.0.0.1" );
LoginResponse response = sessionManagerService.login( request );
String sessionId = response.getSessionID();
```

A Visual Basic code snippet for logging in and then logging out:...

```
Dim loginRequest As New tvSessionMgr.LoginRequest
Dim loginResponse As New tvSessionMgr.LoginResponse
loginRequest.loginType = tvSessionMgr.LoginType.USERNAME
loginRequest.sessionType = tvSessionMgr.SessionType.Traverse
loginRequest.username = "my_user"
loginRequest.password = "my_password"
loginRequest.remoteAddress = "192.168.1.100"
loginResponse = sessionManagerObject.login(loginRequest)
...
Dim logoutRequest As New nvSessionMgr.LogoutRequest
logoutRequest.sessionID = loginResponse.sessionID
sessionManagerObject.logout(logoutRequest)
...
```

Types

```
// Get all available device types
ListDeviceTypesRequest listDeviceTypesRequest = new
ListDeviceTypesRequest();
listDeviceTypesRequest.setSessionId( sessionId );
ListDeviceTypesResponse listDeviceTypesResponse =
typesService.listDeviceTypes(
listDeviceTypesRequest );

// Get all available test sub-types
ListTestTypesRequest listTypesRequest = new ListTestTypesRequest();
listTypesRequest.setSessionId( sessionId );
ListTestTypesResponse listTypesResponse = typesService.listTestTypes(
listTypesRequest );

// Get all available severities
ListSeveritiesRequest listSeveritiesReq = new ListSeveritiesRequest();
listSeveritiesReq.setSessionId( sessionId );
ListSeveritiesResponse listSeveritiesResp =
typesService.listSeverities(listSeveritiesReq) ;
```

Containers, Devices & Tests

```
// Get the status for all containers
GetStatusRequest containerStatusReq = new GetStatusRequest();
containerStatusReq.setSessionId( sessionId );
ObjectFilter containerFilter = new ObjectFilter();
containerStatusReq.setObjectFilter( containerFilter );
containerFilter.setContainerName( "*" );
GetContainerStatusResponse containerStatusResp =
containerService.getContainerStatus( containerStatusReq );

// Get the device status for all devices with 'zyrion' in the name
GetStatusRequest deviceStatusReq = new GetStatusRequest();
deviceStatusReq.setSessionId( sessionId );

// Select only the devices with 'zyrion' in the name
ObjectFilter deviceFilter = new ObjectFilter();
deviceFilter.setDeviceName( "*zyrion*" );
deviceStatusReq.setObjectFilter( deviceFilter );
GetDeviceStatusResponse deviceStatusResp = deviceService.getDeviceStatus(
deviceStatusReq );

// Get 6 hours of historical data for the ping round trip time on web servers
GetHistoricalDataRequest histDataRequest = new GetHistoricalDataRequest();
histDataRequest.setSessionId( sessionId );

// Build the filter
List<String> topNSubTypes = new LinkedList<String>();
topNSubTypes.add( "rtt" );
ObjectFilter testFilter = new ObjectFilter();
testFilter.setDeviceName( "www*" );
testFilter.setTestSubTypes( topNSubTypes );
histDataRequest.setTestFilter( testFilter );
histDataRequest.setStartTime( "6-hours-ago" );
histDataRequest.setEndTime( "now" );
GetHistoricalDataResponse histResp = testService.getHistoricalData(
histDataRequest );
```

Events Service

```
// Get the hourly event distribution for the last 7 days for the
// physical memory usage test on the demo server.
// Only return warning and critical events.
GetEventDistributionRequest getDistReq = new GetEventDistributionRequest();
getDistReq.setSessionId( sessionId );
getDistReq.setStartTimeExp( "7-days-ago" );
getDistReq.setEndTimeExp( "now" );
getDistReq.setGroupingPeriod( GroupingPeriod.HOURS_1 );
getDistReq.setSeverities( new Integer[] { Severity.CRITICAL,Severity.WARNING
} );
getDistReq.setGraphClass( ObjectClass.TEST );
getDistReq.setFilterClass( ObjectClass.TEST );
ObjectFilter criteria = new ObjectFilter();
criteria.setTestName( "Physical Memory Usage" );
criteria.setDeviceName( "demo.zyrion.com" );
getDistReq.setObjectFilter( criteria );
GetEventDistributionResponse getDistResp =
eventService.getEventDistribution( getDistReq );
```

Chapter 9

Traverse CLI (Command Line Interface)

In This Chapter

Overview

The BVE API Command-Line Interface (bveCLI) utility is bundled with **Traverse** 7.0 and provides a convenient method for retrieving information from **Traverse** databases without requiring programming knowledge. The tool can be used to review the configuration of a device, test, container, etc. or create/update various objects or retrieve results/events from one or more DGEs. bveCLI provides numerous advantages over using a Telnet client to connect to the BVE API server, including automatic login, command history recall, inline editing of API command, filter using regular expression, choice of output format, and many others.

Prerequisites

bveCLI uses a BVE API component that is typically installed on the primary/central **Traverse** server. Before the tool can be used, the API server must be started.

Linux/Solaris

Execute the following commands, substituting the proper path for the **Traverse** installation location.

```
su
cd /usr/local/traverse
etc/bveapi.init start
```

Windows

Launch Start > Traverse > Traverse Service Controller and enable the BVE API Server or open a command window and execute the following command.

```
net start nvbveapi
```

General Syntax

bveCLI is installed in the `TRAVERSE_HOME/utils directory`. The tool can be launched using the following parameters:

```
bveCLI.pl [ <server> ] [ <credentials> ] [ <query> ]
```

where `<server>` is specified as

```
--host n.n.n.n --port nnn
```

and credentials are provided as

```
--username USERNAME1 --password PASSWORD1
```

In absence of any server information, bveCLI starts a basic command shell and accepts a limited set of commands.

Sample Session

```
% bveCLI.pl --host 192.168.10.21 --user admin --password secret1
connected to '192.168.10.21:7661'
logged into api server as 'admin'
traverse[192.168.10.21:7661]#
```

Additional query parameters provide advanced functionality and are described in detail in relevant sections below.

Parameter	Short	Description
--config	-c	Load this configuration file instead of looking for .bveapirc under user's home directory
--exec	-x	Execute the specified query in batch mode; requires --host parameter
--input	-i	Execute the queries from specified file in batch mode; requires --host parameter
--output	-o	Save the result from query into specified file; if already exists, it will be overwritten
--format	-t	Result from query should be printed in specified format
--fields	-f	Include only the specified fields in the output; valid only with --exec or --input parameters
--debug	-d	Provide diagnostic details; use multiple times to increase logging detail

Configuration File

bveCLI can load the login credentials for a target BVE API server from a configuration (preferences) file located under the user's home directory. On Linux/Solaris this can be accessed as `$HOME/.bveapirc` while on Windows it is accessible as `%HOME%\.bveapirc`. bveCLI can load a configuration file located at an alternate location using the `--config` parameter. The configuration file has the following general format:

```
[n.n.n.n]
_default=user1
user1=password1
```

where `n.n.n.n` is the fully-qualified domain name or IP address of the BVE server. The configuration file can support multiple sections, each represented with its own BVE server address. The default user ID for a server is specified using the `_default` property. If specified, upon connecting to the server, bveCLI attempts to log in as the specified user. The password for this or other users are specified within the same section as name-value pairs. You may specify multiple credential pairs and the correct password will be looked up when logging into the API server using the corresponding user.

Example: $HOME/.bveapirc

```
[myServer1]
_default=admin
admin=secret1
demo1=letmein
```

```
[yourServer2]
visitor=public1
```

When bveCLI is launched as `bveCLI.pl --host myServer1`, the tool attempts to log into the server as user `admin` using password `secret1` automatically. Alternatively, `bveCLI.pl --host mySserver1 --user demo1` results in automatic login using the password `letmein`. Finally, when launched as `bveCLI.pl --host yourServer2`, the tool establishes a connection to the server but does not attempt to automatically log into it.

Session Management

When launched in interactive mode, bveCLI supports the following commands.

connect <n.n.n.n> [<port>]

Establishes a TCP connection to specified BVE API server. If the port number is not specified, 7661 is used by default. This step can be skipped by specifying the `--host` and `--port` command-line parameters. If a matching entry is found in the configuration file, automatic login will be attempted. Once a connection has been established, the prompt reflects the host name/address and port.

login <username> [<password>]

Logs into the BVE API server using the specified credentials. If the password is omitted, bveCLI attempts to lookup a matching entry from the configuration file. Until a valid credential has been supplied, the prompt displays the user as `unauth`.

exit

Logs out of the API server and close TCP connection.

Sample Session

```
% bveCLI.pl
traverse> connect 192.168.10.21
connected to '192.168.10.21:7661'
traverse(noauth)> login admin secret1
logged into api server as 'admin'
traverse[192.168.10.21:7661]# quit
```

```
% bveCLI.pl --host 192.168.10.21 --user admin --password secret1
connected to '192.168.10.21:7661'
logged into api server as 'admin'
traverse[192.168.10.21:7661]# quit
```

Running A Query

bveCLI supports all standard BVE API commands with corresponding parameters, as outlined in the **BVE FlexAPI Protocol Reference** *(page 3)*. A command issued on bveCLI is transparently executed on the remote server and the response from the server is parsed/analyzed. For configuration commands that only return a success/failure response, no output is presented unless the command was not successful. A command that returns configuration/performance data (eg. device.list, location.list) is presented in a readable format.

Sample Session

```
traverse[192.168.10.21:7661]# device.list "devicename=Cisco*"
.----------------------------------------------------------.
|                   Number of Items: 2                     |
+--------------------------+-------------------------------+
| Parameter                | Value                         |
+--------------------------+-------------------------------+
|              accountname | Zyrion, Inc.                  |
|      accountserialnumber | 49                            |
|                  address | 192.168.10.250                |
|           agentbatchmode | 1                             |
|           agentcommunity | secret1                       |
|                agentport | 161                           |
|             agentversion | 2                             |
|                clearonok | false                         |
|                  comment | WAN Router/Default Gateway    |
|      configbackupenabled | true                          |
|    configbackupfrequency | 360                           |
|               devicename | Cisco Router                  |
|               devicetype | IP Router                     |
|                  dgename | localhost                     |
| flappreventionwaitcycles | 0                             |
|               isimported | false                         |
|               isreadonly | false                         |
|              issuspended | false                         |
|             locationname | Corporate Office              |
|                    model | 3620                          |
|              parentnames |                               |
|       rediscoveryenabled | false                         |
|             serialnumber | 220070                        |
|            showonsummary | true                          |
|              smartnotify | true                          |
|             tag1 (tag 1) | IT                            |
|             tag2 (tag 2) |                               |
|             tag3 (tag 3) |                               |
|             tag4 (tag 4) |                               |
|             tag5 (tag 5) |                               |
|              v3authproto | 1                             |
|              v3privproto | 1                             |
|                   vendor | Cisco Systems                 |
+--------------------------+-------------------------------+
|              accountname | Zyrion, Inc.                  |
|      accountserialnumber | 49                            |
|                  address | 192.168.14.1                  |
|                clearonok | false                         |
|                  comment |                               |
|      configbackupenabled | false                         |
|               devicename | Cisco UCS Platform            |
|               devicetype | Other/Generic Device          |
|                  dgename | localhost                     |
| flappreventionwaitcycles | -1                            |
|               isimported | false                         |
|               isreadonly | false                         |
|              issuspended | false                         |
|             locationname | Corporate Office              |
|                    model |                               |
|              parentnames | Primary Distribution Switch  |
|       rediscoveryenabled | false                        |
|             serialnumber | 2400080                      |
|            showonsummary | true                         |
|              smartnotify | true                         |
|             tag1 (tag 1) |                              |
|             tag2 (tag 2) |                              |
|             tag3 (tag 3) |                              |
|             tag4 (tag 4) |                              |
|             tag5 (tag 5) |                              |
|                   vendor |                              |
'--------------------------+------------------------------'
```

```
traverse[192.168.10.21:7661]# test.list "devicename=Cisco Router",
"testname=Proc*Memory*"
.-----------------------------------------------------------.
|                    Number of Items: 1                     |
+--------------------------+--------------------------------+
| Parameter                | Value                          |
+--------------------------+--------------------------------+
|              accountname | Zyrion, Inc.                   |
|               actionname | None                           |
|           agentbatchmode | 1                              |
|           agentcommunity | secret1                        |
|                agentport | 161                            |
|             agentversion | 2                              |
|        criticalthreshold | 95                             |
|               devicename | Cisco Router                   |
| flappreventionwaitcycles | -1                             |
|                 interval | 600                            |
|              issuspended | false                          |
|                 maxvalue | 47536288                       |
|         resultmultiplier | 1.0                            |
|   resultprocessdirective | 1                              |
|             schedulename | Default Schedule               |
|             serialnumber | 250005                         |
|  shadowcriticalthreshold | 95                             |
|   shadowwarningthreshold | 85                             |
|             slathreshold | 76                             |
|                  snmpoid | .1.3.6.1.4.1.9.9.48.1.1.1.5.1  |
|                  subtype | cisco_memfree                  |
|               suppressed | false                          |
|                 testname | Processor Memory Util          |
|                 testtype | snmp                           |
|            thresholdtype | 1                              |
|      timebasedthresholds | false                          |
|                    units | %                              |
|              v3authproto | 1                              |
|              v3privproto | 1                              |
|         warningthreshold | 85                             |
'--------------------------+--------------------------------'
```

```
traverse[192.168.10.21:7661]# device.create "devicename=TEST1",
"location=Corporate Office", "devicetype=windows"
ERROR: Missing required fields '"address=<value>"', and
'"locationName=<value>"'.
```

```
traverse[192.168.10.21:7661]# device.create "devicename=TEST1",
"address=127.0.0.1", "locationname=Corporate Office", "devicetype=windows"
```

```
traverse[192.168.10.21:7661]# device.list "devicename=TEST1"
.-----------------------------------------------.
|              Number of Items: 1               |
+---------------------------+-------------------+
| Parameter                 | Value             |
+---------------------------+-------------------+
|               accountname | Zyrion, Inc.      |
|       accountserialnumber | 49                |
|                   address | 127.0.0.1         |
|                 clearonok | false             |
[...]
traverse[192.168.10.21:7661]# device.delete "devicename=TEST1"
```

```
traverse[192.168.10.21:7661]#
```

Search Filter

Beyond the search criteria supported by different BVE API commands, bveCLI provides additional filtering capabilities. Commands executed on bveCLI can be piped through a grep filter. The filter supports Perl5 compliant regular expressions and is applied against the raw output from the remote server in case insensitive manner before parsed by bveCLI.

Sample Session

```
traverse[192.168.10.21:7661]# device.status "status=warning"
.----------------------------------------.
|           Number of Items: 3           |
+--------------+-------------------------+
| Parameter    | Value                   |
+--------------+-------------------------+
|   devicename | Exchange Server 2007    |
| serialnumber | 1100004                 |
|       status | Warning                 |
+--------------+-------------------------+
|   devicename | Shipping Floor Printer  |
| serialnumber | 280247                  |
|       status | Warning                 |
+--------------+-------------------------+
|   devicename | Database Server         |
| serialnumber | 540012                  |
|       status | Warning                 |
'--------------+-------------------------'
```

```
traverse[192.168.10.21:7661]# device.status "status=warning" | grep server
.-------------------------------------.
|          Number of Items: 2         |
+--------------+----------------------+
| Parameter    | Value                |
+--------------+----------------------+
|   devicename | Exchange Server 2007 |
| serialnumber | 1100004              |
|       status | Warning              |
+--------------+----------------------+
|   devicename | Database Server      |
| serialnumber | 540012               |
|       status | Warning              |
'--------------+----------------------'
```

Batch Processing

bveCLI can also be used in an non-interactive manner for performing quick queries against the BVE API server. When executed with the `--exec` parameter, bveCLI executes the specified query and logs out immediately. In this case, the `--host` parameter must be specified along with suitable login credentials provided in a command line or configuration file. The search filter can be specified in batch mode, similar to the **interactive mode** *(page 90)*.

Sample Session

```
% bveCLI.pl --host 192.168.10.21 --user admin --exec 'container.status
"servicename=S*"'
.------------------------------------.
|         Number of Items: 3         |
+--------------------+---------------+
| Parameter          | Value         |
+--------------------+---------------+
| parentserialnumber | 1770035       |
|       serialnumber | 1770023       |
|        servicename | Sunnyvale     |
|             status | Ok            |
+--------------------+---------------+
| parentserialnumber | 1770035       |
|       serialnumber | 1770029       |
|        servicename | San Francisco |
|             status | Critical      |
+--------------------+---------------+
| parentserialnumber | 1770035       |
|       serialnumber | 2890000       |
|        servicename | San Antonio   |
|             status | Ok            |
'--------------------+---------------'
```

```
% bveCLI.pl --host 192.168.10.21 --exec 'container.status "servicename=S*" |
grep critical'
.--------------------------------------.
|         Number of Items: 1           |
+---------------------+----------------+
| Parameter           | Value          |
+---------------------+----------------+
| parentserialnumber  | 1770035        |
|       serialnumber  | 1770029        |
|        servicename  | San Francisco  |
|             status  | Critical       |
'---------------------+----------------'
```

In order to execute multiple commands, use `--input` parameter instead. This parameter requires the location of a text file containing one or more valid API commands on each line.

Sample Session

```
% cat /tmp/commands.txt
container.status "servicename=S*"
device.status "status=warning"
```

```
% bveCLI.pl --host 192.168.10.21 --user admin --input /tmp/commands.txt
.-------------------------------------.
|         Number of Items: 3          |
+--------------------+----------------+
| Parameter          | Value          |
+--------------------+----------------+
| parentserialnumber | 1770035        |
|       serialnumber | 1770023        |
|        servicename | Sunnyvale      |
|             status | Ok             |
+--------------------+----------------+
| parentserialnumber | 1770035        |
|       serialnumber | 1770029        |
|        servicename | San Francisco  |
|             status | Critical       |
+--------------------+----------------+
| parentserialnumber | 1770035        |
|       serialnumber | 2890000        |
|        servicename | San Antonio    |
|             status | Ok             |
'--------------------+----------------'
.----------------------------------------.
|           Number of Items: 3           |
+--------------+-------------------------+
| Parameter    | Value                   |
+--------------+-------------------------+
|   devicename | Exchange Server 2007    |
| serialnumber | 1100004                 |
|       status | Warning                 |
+--------------+-------------------------+
|   devicename | Shipping Floor Printer  |
| serialnumber | 280247                  |
|       status | Warning                 |
+--------------+-------------------------+
|   devicename | Database Server         |
| serialnumber | 540012                  |
|       status | Warning                 |
'--------------+-------------------------'
```

Field Selection

In batch mode, bveCLI allows selection of specific configuration/output fields using the `--fields` parameter. When used, the output only includes the specified fields.

Sample Session

```
% bveCLI.pl --host 192.168.10.21 --exec 'device.list "devicename=Cisco*"'
--fields
devicename,address,configbackupenabled,devicetype,invalidfield,serialnumbe
r
.-----------------------------------------------------.
|                  Number of Items: 3                 |
+---------------------+-------------------------------+
| Parameter           | Value                         |
+---------------------+-------------------------------+
|             address | 67.21.1.74                    |
| configbackupenabled | true                          |
|          devicename | Cisco Router                  |
|          devicetype | IP Router                     |
|        serialnumber | 220070                        |
+---------------------+-------------------------------+
|             address | 192.168.14.1                  |
| configbackupenabled | false                         |
|          devicename | Cisco UCS Platform            |
|          devicetype | Other/Generic Device          |
|        serialnumber | 2400080                       |
+---------------------+-------------------------------+
|             address | 192.168.14.2                  |
| configbackupenabled | false                         |
|          devicename | Cisco Unified Communications |
|          devicetype | Linux/Other Unix              |
|        serialnumber | 2400399                       |
'---------------------+-------------------------------'
```

Output Format

By default bveCLI uses a tabular output format that is suitable for operational use. The tool supports additional output formats—XML, JSON, CSV—that can be used for integration with in-house and third-party tools. The output format can be selected using the `--format` parameter.

Sample Session

```
% bveCLI.pl --host 192.168.10.21 --exec 'device.list "devicename=Cisco*"'
--fields
devicename,address,configbackupenabled,devicetype,invalidfield,serialnumbe
r --format json
```

```
{
   "api-response" : {
      "data" : {
         "object" : [
            {
               "address" : "67.21.1.74",
               "configbackupenabled" : "true",
               "devicename" : "Cisco Router",
               "devicetype" : "IP Router",
               "serialnumber" : "220070"
            },
            {
               "address" : "192.168.14.1",
               "configbackupenabled" : "false",
               "devicename" : "Cisco UCS Platform",
               "devicetype" : "Other/Generic Device",
               "serialnumber" : "2400080"
            },
            {
               "address" : "192.168.14.2",
               "configbackupenabled" : "false",
               "devicename" : "Cisco Unified Communications",
               "devicetype" : "Linux/Other Unix",
               "serialnumber" : "2400399"
            }
         ]
      },
      "status" : {
         "code" : "203",
         "error" : "false",
         "message" : "request accepted, records returned: 3"
      }
   }
}
```

```
% bveCLI.pl --host 192.168.10.21 --exec 'device.list "devicename=Cisco*"'
--fields
devicename,address,configbackupenabled,devicetype,invalidfield,serialnumbe
r --format xml
```

```
<?xml version='1.0' standalone='yes'?>
<api-response>
  <data>
    <object>
      <serialnumber>220070</serialnumber>
      <address>67.21.1.74</address>
      <configbackupenabled>true</configbackupenabled>
      <devicename>Cisco Router</devicename>
      <devicetype>IP Router</devicetype>
    </object>
    <object>
      <serialnumber>2400080</serialnumber>
      <address>192.168.14.1</address>
      <configbackupenabled>false</configbackupenabled>
      <devicename>Cisco UCS Platform</devicename>
      <devicetype>Other/Generic Device</devicetype>
    </object>
    <object>
      <serialnumber>2400399</serialnumber>
      <address>192.168.14.2</address>
      <configbackupenabled>false</configbackupenabled>
      <devicename>Cisco Unified Communications</devicename>
      <devicetype>Linux/Other Unix</devicetype>
    </object>
  </data>
  <status>
    <code>203</code>
    <error>false</error>
    <message>request accepted, records returned: 3</message>
  </status>
</api-response>
```

```
% bveCLI.pl --host 192.168.10.21 --exec 'device.list "devicename=Cisco*"'
--fields
devicename,address,configbackupenabled,devicetype,invalidfield,serialnumbe
r --format csv
```

```
# serialnumber, address, configbackupenabled, devicename, devicetype
"220070", "67.21.1.74", "true", "Cisco Router", "IP Router"
"2400080", "192.168.14.1", "false", "Cisco UCS Platform", "Other/Generic
Device"
"2400399", "192.168.14.2", "false", "Cisco Unified Communications",
"Linux/Other Unix"
```

For JSON/XML formats, the objects are returned as an array with separate elements indicating their success/failure status. For CSV format, the first row is prefixed with # symbol and indicates the contents of each column.

Future Enhancements

- Interactive password input should mask user entry
- Pagination support using "| more"
- Interactive field selection using "| cut"
- Tab completion works for top level command but not parameters

Index