



NetVigil Perl API

25.1 Introduction

The NetVigil 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.

25.2 NetVigil::ExternalData - EDF API

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

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

25.2.1 Methods

new

Create a new `NetVigil::ExternalData` object

```
use NetVigil qw(ExternalData);

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

This is the constructor for `NetVigil::ExternalData` objects. A new object is returned on success. The `$login_id` and `$login_password` parameters can be omitted and specified during `Login` method. No connection is made to the remote host when this method is called. Only

the object is created with remote host address and port information. The new object returned is given the following defaults in the absence of corresponding named arguments:

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

GetErrorMsg

Retrieve error information from last operation

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

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

Login

Log into NetVigil EDF server.

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

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

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

Logout

Log out of the NetVigil EDF server.

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

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

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

InsertResult

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

Refer to Chapter 24, “External Data Feed (EDF) Reference” for explanations of the parameters and valid values.

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

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

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

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

25.3 NetVigil::Message - ISM API

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

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

25.3.1 Methods

new

Create a new NetVigil::Message object

```
use NetVigil qw(Message);

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

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

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

GetErrorMsg

Retrieve error information from last operation

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

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

Login

Log into NetVigil ISM server.

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

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

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

Logout

Log out of the NetVigil ISM server.

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

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

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

InsertMessage

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

or

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

The first form of the method will insert a message specific for the device (and optionally test) specified into the system for further processing. The second method will force the system to match the message against all configured regular expression patterns and if there is a match,

appropriate severity will be set and actions will be triggered. Refer to Section 7.9.2, “Input Stream Monitor (ISM)” on page 99 for explanations of the parameters and valid values for the first method.

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

Example: Connect, log in, insert message, log out

```
.....  
use NetVigil qw(Message);  
my $obj = new NetVigil::Message(Host=>"localhost");  
my $return_value = $obj->  
    Login(User=>"ismuser",Password=>"fixme") ||  
    die "ERROR: ", $obj->GetErrorMsg, "\n";  
$obj->InsertMessage(deviceName=>"my_server",  
                    deviceAddr=>"192.168.200.50",  
                    dateTime=>time,  
                    severity=>'ok',  
                    Message=>"this is a message") ||  
    print "ERROR: ", $a->GetErrorMsg, "\n";  
$obj->Logout;  
.....
```

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

25.4 NetVigil::Provisioning - BVE API

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

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

The detailed list of commands and parameters expected by the BVE socket server is detailed in Chapter 23, “BVE FlexAPI Protocol Reference”.

25.4.1 Methods

new

Create a new `NetVigil::Provisioning` object

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

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

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

GetErrorMsg

Retrieve error information from last operation.

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

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

GetResultCount

Return the number of objects in the result buffer.

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

This method provides a count of the number of objects that were found in the result of an earlier `List<object>` method (see “`CreateX`, `ListX`, `UpdateX`, `DeleteX`, `SuspendX`, `ResumeX`” on page 325). Note that if the result of the `List<object>` method returned results in bulk format (e.g. `ListResult` or `ListEvent`), this method will always return 0 since the results cannot be accessed using the `GetResultRef` method. Instead, look at the size of the array returned for the `GetResultSet` method.

GetResultRef

ReturnPointer to search result buffer.

```

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

```

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

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

```

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

```

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

GetResultSet

Return results of a bulk search.

```

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

```

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

Login

Log into NetVigil socket server.

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

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

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

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

Logout

Log out of the NetVigil socket server.

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

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

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

CreateX, ListX, UpdateX, DeleteX, SuspendX, ResumeX

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

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

- AdminGroup
- UserClass
- Department

- User
- Location
- DGE
- Device
- Test
- Action

The parameters for each object and method combination are different. Refer to Chapter 23, “BVE FlexAPI Protocol Reference” 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 GetErrorMsg method. On success non-zero return value is provided. Results for ListX methods are stored in an internal array and accessed using GetResultRef method.

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

25.5 Examples

Connect, log in, create a DGE, log out

```

.....
use NetVigil qw(Provisioning);
my $obj = new NetVigil::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, 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:

Create a device for a DGE, then list devices

```
.....
use NetVigil qw(Provisioning);

my $obj = new NetVigil::Provisioning(Host=>"localhost");
my $return_value = $obj->
    Login(User=>"admin", Password=>"changeme") ||
    die "ERROR: ", $obj->GetErrorMsg, "\n";

my %param = ();
$params{deviceName} = "my test device";
$params{address} = "192.168.200.50";
$params{locationName} = "Local Network";
$params{snmpCid} = "public";
$params{comment} = "my workstation";
$params{devicetype} = "unix";
$obj->CreateDevice(%param) || die "ERROR: ", $obj->GetErrorMsg,
"\n";

$obj->ListDevice(deviceName=>'my *');
if ($obj->GetResultCount) {
my $result_ref = $obj->GetResultRef();
foreach my $serial_num (keys %{ $result_ref }) {
    print "device with serial number ${serial_num} ..\n";
    foreach my $object_param (keys %{ $result_ref->{$serial_num} })
    {
        $param_value = $result_ref->{$serial_num}->{$object_param};
        print "\t\t${object_param} = ${param_value}\n";
    }
}
}

$obj->Logout;
```

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

Finding tests without Actions assigned

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

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

```

my %DEVICE_LIST = ();
my $RESULT_REF = $BVE->GetResultRef();
foreach my $device_serial (keys %{ $RESULT_REF }) {
    my $device_name =
        $RESULT_REF->{$device_serial}->{devicename};
        $DEVICE_LIST{$device_serial} = $device_name;
    }
## now scan through tests on each device
foreach my $device_serial (sort keys %DEVICE_LIST) {
    $BVE->ListTest(deviceName=>$DEVICE_LIST{$device_serial},
testName=>'*');
    $RESULT_COUNT = $BVE->GetResultCount();
    next unless ($RESULT_COUNT);
    $RESULT_REF = $BVE->GetResultRef();
    foreach my $test_serial (keys %{ $RESULT_REF }) {
        my $action_profile =
            $RESULT_REF->{$test_serial}->{actionname};
        my $test_name = $RESULT_REF->{$test_serial}->{testname};
        next unless (uc($action_profile) eq "NONE");
        &info("device = $DEVICE_LIST{$device_serial} ; test =
\'$test_name\");
        } # foreach test
    } # foreach device
$BVE->Logout;
.....

```

Creating a custom SNMP test

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

```

my $obj = new NetVigil::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

```


