



## Plugin Monitors

### 27.1 Overview

The plugin monitor functionality in NetVigil 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 NetVigil 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 `NETVIGIL_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.

### 27.2 Adding A New Test Type

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

When creating new (plugin) monitors, you will need to create a unique test type and sub-type for that monitor and provide various default values and other parameters. The entries in `NETVIGIL_HOME/etc/TestTypes.xml` or other directories **should not** be edited (unless you are instructed to edit them by Fidelia support) as it may adversely affect or cause failure of NetVigil components. Any

changes made to these directories may also be lost when a new version of NetVigil is installed. All user customizations are expected to be placed in `NETVIGIL_HOME/plugin` and its subdirectories.

### Sample TestTypes.xml entry

```

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

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

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

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

```

The testtype element includes the following child elements:

**Table 27.1** XML TestType element

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 <code>network</code> , <code>system</code> , and <code>application</code> .
subType	This is a string that uniquely identifies the testtype to the NetVigil 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: <code>true</code> <code>false</code> or <code>static</code> If the value is <code>true</code> , then the status being tested becomes more critical as the test value rises. When the value is <code>static</code> , you can set discrete threshold values for warning and critical.
defaultWarningThreshold	This is the default end-user warning threshold for this test type. If the <code>severityAscendsWithValue</code> is <code>'static'</code> , 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.

**Table 27.1** XML TestType element

child element	description
showAsGroup	Group tests of the same sub-type together in the web app 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	<p>This element defines a specific attribute for the test. A testtype can have 0 or more testfields. Each testfield should have the following child elements:</p> <p><code>fieldName</code> - This will be used as key for the field value when it's passed to the test.</p> <p><code>fieldDisplayName</code> - A user friendly name for the field that will be used by the web application when creating or updating tests.</p> <p><code>isRequired</code> - This element indicates whether or not the a value is required to be given for the field when creating or updating the test.</p> <p><code>isPassword</code> - This indicates whether or not the field is a password field. The web application will ask for verification of password fields when creating or updating a test.</p> <p><code>defaultValue</code> - A default value that will be presented to a user when creating the test.</p>

## 27.3 Creating A New Plugin Java Monitor

### 27.3.1 Overview

NetVigil 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.

### 27.3.2 Configuration File Format

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

#### Weather information plugin test descriptor

```
.....  
<monitor type="weather" pluginType="java"  
resource="com.weatherwatchers.netvigilplugin.WeatherPlugin">  
  <testtype>  
    <displayName>Current Temperature</displayName>  
    <displayCategory>application</displayCategory>  
    <subType>temperature</subType>  
    <units>degrees C</units>  
    <severityAscendsWithValue>true</severityAscendsWithValue>  
    <defaultWarningThreshold>100</defaultWarningThreshold>  
    <defaultCriticalThreshold>120</defaultCriticalThreshold>  
    <shadowWarningThreshold>100</shadowWarningThreshold>  
    <shadowCriticalThreshold>120</shadowCriticalThreshold>  
    <slaThreshold>120</slaThreshold>  
    <testInterval>180</testInterval>  
    <testField>  
      <fieldName>city</fieldName>  
      <fieldDisplayName>City</fieldDisplayName>  
      <isRequired>true</isRequired>  
      <isPassword>false</isPassword>  
      <defaultValue>Muskogee</defaultValue>  
    </testField>  
    <testField>  
      <fieldName>state</fieldName>  
      <fieldDisplayName>State/Province</fieldDisplayName>  
      <isRequired>true</isRequired>  
      <isPassword>false</isPassword>  
      <defaultValue>OK</defaultValue>  
    </testField>
```

```

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

```

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

**Table 27.2** Monitor element attributes

attribute	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 <code>java</code> .
resource	This is the name of the resource of that should be used to do the tests. For a <code>plugintype</code> of <code>java</code> , this should be the fully qualified name of a Java class file that implements the <code>NetvigilPlugin</code> interface.

The configuration file also requires a `testType` definition as described above on page 339. You should make a different xml test descriptor for each type of monitor you want to create.

### 27.3.3 Writing The Plugin Class

Your plugin class should be able to be run with the Sun 1.4 JVM. 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 NetVigil socket interface. Once created, the plugin tests are stored in the provisioning database, with each of the `testField` values for that test, with the `fieldName` as a key. The DGE loads the tests from the database, and after it has determined that the testing interval has passed, adds the test to a test queue, indicating that the test should be run.

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

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

### 27.3.4 Setting up 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 `NETVIGIL_HOME/plugin/monitors`. Make a directory in a `NETVIGIL_HOME/plugin/monitors` called `<type>/lib`, where `<type>` is

the type of monitor in your XML test descriptor. So, for the weather example described in “Weather information plugin test descriptor” on page 341, you should create a directory called `NETVIGIL_HOME/plugin/monitors/weather/lib`. Place the `.jar` file you just created in the `lib` directory. If NetVigil is installed in a distributed environment (multiple hosts), the plugin package and test descriptor file should be installed on each host running NetVigil.

### 27.3.5 Provisioning Plugin Tests

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

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

## 27.4 Creating A New Plugin Script Monitor

### 27.4.1 Configuration File Format

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

```
<monitor type="weather" plugintype="script">
```

```

<!--
Insert a testType element here.
-->

<script type="weather" subType="temperature">
  <rootScript>gettemp.pl</rootScript>
  <timeout>10</timeout>
  <parameters>--country=${country} --state=${state}
  --city=${city} </parameters>
</script>
</monitor>
    
```

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

**Table 27.3**

attribute	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 script plugin monitor, this parameter should be set to <code>script</code> .

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

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

The next two child elements are fairly straightforward. The `rootScript` element gives the name of the script to run, and the `timeout` element gives the maximum number of seconds to wait for the script. You should give a timeout of less than 60 seconds for your script, so that the

NetVigil monitor running your script can return in a timely manner if your script hangs for some reason. Timeout values of zero or less will be interpreted as a 60-second timeout.

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

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

```

#{device_name}           #{device_serial_number}
#{device_address}       #{device_model}
#{device_vendor}        #{device_type}
#{device_snmp_cid}      #{device_snmp_version}
#{device_location}      #{test_name}
#{test_serial_number}   #{test_user_warning_threshold}
#{test_user_critical_threshold}  #{test_shadow_warning_
                                threshold}
#{test_shadow_critical_threshold}  #{test_sla_threshold}
#{test_type}            #{test_sub_type}
#{test_units}
  
```

When NetVigil 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, NetVigil 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.

## 27.4.2 Writing The Plugin Script

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

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

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

## 27.4.3 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 will be in warning state, critical is 4,8,9,10 and 99. Everything else is OK.

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

```

#!/bin/sh
#if [ -f "/tmp/atmosphere.dat" ]; then
cat /tmp/atmosphere.dat
else
echo 0
fi

```

3. Now restart the web application, and then provision the test:
  - a) log in as end-user
  - b) click on `manage->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

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

```

## 27.5 Extending the Message Handler

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

As an example, you can add new log files to be monitored and a trap handler listening on port 2162 by creating the following two files in the `NETVIGIL/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 Section 7.5, “Adding Rulesets” on page 89

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

