Methods of monitoring processes with Zenoss

June 2009

Jane Curry

Skills 1st Ltd

www.skills-1st.co.uk

Jane Curry
Skills 1st Ltd
2 Cedar Chase
Taplow
Maidenhead
SL6 0EU
01628 782565

jane.curry@skills-1st.co.uk

Synopsis

This paper discusses three possible methods for performing process monitoring;

- Using the process monitoring capabilities of the net-SNMP agent
- Using SSH to access a device and run local commands, Nagios-style plugins or Zenoss plugins
- Using Zenoss's zenprocess daemon

Each of these methods will be examined, including an in-depth discussion on the various different types of plugin that Zenoss can utilise and their strengths and weaknesses. Examples and screenshots are provided.

In addition to monitoring processes, the options for rectifying failed processes will be explored. This can be driven by the Zenoss events subsystem so examples are given to generate events from each of the process monitoring techniques.

A third element of monitoring processes is to collect performance data for use in graphs and threshold-generated events. Examples of performance data collection templates are included for each of the SSH-based methods.

It is assumed that the reader is familiar with basic SNMP concepts and with simple SNMP configuration parameters. It is also assumed that the reader is familiar with setting up communications using SSH.

This paper was written based on stack-built Zenoss Core 2.3.3 on SLES 10.

Table of Contents

1 Overview of process management

1.1 Defining "process management" requirements

"Process management" can encompass a wide variety of interpretations:

1. Monitoring for processes on Unix / Linux, effectively using output from some invocation of the *ps* command
2. Monitoring processes as defined by entries in Task Manager on a Windows system
3. Monitoring for a single occurrence of a simple process name (eg. named)
4. Monitoring for full pathname of a command (eg. /usr/sbin/named)
5. Monitoring the arguments of a command
6. Monitoring for minimum and/or maximum numbers of occurrences of a process
7. "Alerting" on process failure and recovery
8. Automatic recovery from process failure

With the exception of monitoring Windows processes and services, each of these requirements will be considered against each of the monitoring techniques discussed.

Windows services can be monitored by Zenoss's zenwin daemon and processes (ie. programs that do not run as Windows services but do appear in the Windows Task Manager) can be monitored using the standard Zenoss zenprocess daemon, provided the target supports the SNMP Host Resources MIB. Thus some of the details in this paper are also applicable to Windows targets.

1.2 Methods for monitoring Unix / Linux processes

Ultimately, process monitoring for Unix / Linux systems comes from some form of running the *ps* command. Typically this will be achieved either through SNMP or through SSH.

1.2.1 Native SNMP access to process information

Most Linux distributions use the net-SNMP agent and net-SNMP is also available for proprietary Unix implementations. This paper will assume the presence of net-SNMP agents.

net-SNMP itself provides a number of options for retrieving process information:

- Host Resources MIB (RFC 2790 supercedes RFC 1514)
- net-SNMP process table support from the UCD-SNMP-MIB (net-SNMP used to be UCD snmp)

No form of monitoring is truly “agentless” but since most Operating Systems do provide SNMP, then management by SNMP is fairly close to agentless – once the agent has been configured it should continue to deliver information to a management station.

There are three versions of SNMP (V1, V2c and V3) where V1 and V2c have very little authentication or encryption as part of the protocol, but SNMP V3 can provide both. Obviously SNMP V3 will have a greater performance overhead than the earlier versions.

1.2.2 Using SSH to gain process information

Secure Shell (SSH) can be thought of as another “agentless” method for accessing information. As with SNMP, SSH tends to be supported as standard by most Operating Systems and will operate without intervention once configured.

SSH management solutions tend to be “heavier” in resources. Encryption will be enforced at source, destination and across the network. SSH can permit any script to be run at the managed device so it can be as intensive and comprehensive as required; thus an SSH solution can potentially address all the process management requirements detailed above.

1.2.3 Using Zenoss's zenprocess daemon to monitor process information

Zenoss provides the zenprocess daemon to query the availability and performance of processes on remote devices. Fundamentally, zenprocess makes use of the Zenoss HRSWRunMap data collector which relies on the Host Resources SNMP MIB at the target.

Processes are configured from the main left-hand *Processes* menu. One automatic advantage of using zenprocess is that, in addition to monitoring for the presence of a process, it will also create graphs of that process's CPU, memory usage, and the number of instances of the process (count).

2 Native net-SNMP process management

Strictly, an SNMP agent is only required to support MIB-2 (which largely provides network information); however, many SNMP agents support extra Management Information Bases (MIBs) as standard, and, in particular, many support the Host Resources MIB, a generic MIB that provides system information about a device. The net-SNMP agent can have support for other MIB extensions, such as the process table of the UCD-SNMP-MIB and the DisMan Event MIB, in addition to the Host Resources MIB.

Note that later versions of the net-SNMP agent tend to be distributed with support for many extensions already compiled in, but older versions may not have all the extra extensions; in this case, you may need to get the source of the net-SNMP agent and rebuild it. To find out what your net-SNMP agent supports, run one of the following:

- `net-snmpconfig   snmpd-module-list`
- `snmpd -Dmib_init -H     (needs root privilege)`

To read MIB information from an SNMP agent,the snmpwalk command is a useful testing tool. for example:

- `snmpwalk -v 1 -c public zen232 system`
  - uses SNMP V1 with a community name of public to GET the system table from the machine zen232
- `snmpwalk -v 3 -a MD5 -A fraclmyea -l authNoPriv -u jane2 zen232 system`
  - uses SNMP V3 with MD5 authentication, passphrase fraclmyea, and user jane2 to GET the system table from machine zen232

Obviously, the agent on the target host must have been configured to permit this access, in its snmpd.conf file.

2.1 Host Resources MIB

The Host Resources MIB defined in RFC 1514 and updated by RFC 2790 defines many standard MIB values for monitoring the “health” of a system, including tables for CPU, memory, swap, storage, devices, installed software, running software and the performance of running software.

The hrSWRunTable contains an entry for each distinct piece of software that is running or loaded into physical or virtual memory in preparation for running. This includes the host's operating system, device drivers, and applications. hrSWRunTable consists of a sequence of hrSWRunEntry objects defined as follows:

```
HrSWRunEntry ::= SEQUENCE {
          hrSWRunIndex       Integer32,
          hrSWRunName        InternationalDisplayString,
          hrSWRunID          ProductID,
          hrSWRunPath        InternationalDisplayString,
          hrSWRunParameters  InternationalDisplayString,
          hrSWRunType        INTEGER,
          hrSWRunStatus      INTEGER
      }
```

Typically:

- hrSWRunIndex is the process (PID) eg. 3555
- hrSWRunName is the short name of the process eg. named
- hrSWRunID always seems to be zeroDotZero

- hrSWRunPath is the full pathname eg. /usr/sbin/named
- hrSWRunParameters are the parameters to the command (if any), eg. -t /var/lib/named -u named (**Note** that long lines get truncated!)
- hrSWRunType is generally an application denoted by the integer value of 4
- hrSWRunStatus typically is runnable (2) though at least one process should have a status of running (1)

If multiple instances of a process are running then each is reported, with the process being the differentiator.

The hrSWRunPerf table entry has 2 objects for CPU and memory:

```
HrSWRunPerfEntry ::= SEQUENCE {
            hrSWRunPerfCPU          Integer32,
            hrSWRunPerfMem          KBytes
    }
```

“CPU” is described as “the number of centi-seconds of the total system's CPU resources consumed by this process. Note that on a multi-processor system, this value may increment by more than one centi-second in one centi-second of real (wall clock) time.”

“Memory is defined as “the total amount of real system memory allocated to this process."

The index for both CPU and memory is again the process .

Thus, the Host Resources MIB satisfies requirements 1, 3, 4 and 5 above (monitoring for a process, monitoring the full pathname and monitoring the arguments). Multiple occurrences of a process are reported but there is no simple way to specify *how many* processes should be running.

To examine the Host Resources process information on a target device using SNMP V1 and a community name of public, use:

- `snmpwalk -v 1 -c public zen233 hrSWRunTable`

2.2 Process table of UCD-SNMP-MIB

The net-SNMP agent has become the ubiquitous SNMP agent for Linux and is available for many other systems. It evolved from the University of California Davis (UCD) SNMP agent which had some useful private MIB extensions, including process monitoring. The prTable of the UCD-SNMP-MIB allows specification of a process name (the short name as reported by *ps -acx*) and a maximum and minimum number of occurrences of the process. If the number of processes is less than MIN or greater than MAX, then the corresponding prErrorFlag instance will be set to 1, and a suitable description message reported via the prErrMessage instance. **Note**: This situation will **not** automatically trigger a trap to report the problem - see the DisMan Event MIB section later. The syntax within the snmpd.conf file is:

```
proc named 1  1
proc vmware-vmx 3  4
```

There should be precisely one occurrence of the named process running and at least 3 but no more than 4 occurrences of vmware-vmx.

Optionally, snmpd.conf can also specify a command to run to attempt to fix the problem. This is defined with a procfix line, for example:

```
procfix named /etc/init.d/named start
```

Note that a procfix line must come after the related proc statement. The procfix command will **not** be run automatically. It is only run when the corresponding prErrFix MIB value is set from 0 to 1.

The prTable in the UCD-SNMP-MIB is defined as follows:

```
PrEntry ::= SEQUENCE {                               Index Number
    prIndex             Integer32,                      1
    prNames             DisplayString,                  2
    prMin               Integer32,                      3
    prMax               Integer32,                      4
    prCount             Integer32,                      5
    prErrorFlag         UCDErrorFlag,                   100
    prErrMessage        DisplayString,                  101
    prErrFix            UCDErrorFix,                    102
    prErrFixCmd         DisplayString                   103
}
```

Note that the index numbers for this sequence are not consecutive (see right-hand column). For example, the Object Identifier (OID) for the 5th instance in the process table for prErrorFlag would be *.1.3.6.1.4.1.2021.2.1.100.5*, where *.1.3.6.1.4.1.2021* gets you to ucdavis, the next *.2.1* gets you to prTable.prEntry, *.100* is the prErrorFlag and the final *.5* is the instance denoting the 5th process entry in the table.

Typically:

- The prIndex field is simply an increasing number to index into the process table, starting at 1.
- prNames is the short name of the process eg. vmware-vmx
- The prErrorFlag is set to 1 if the count value exceeds max or is less than min

- prErrMessage reflects a suitable error message if prErrorFlag=1. For example, “Too few vmware-vmx running (# = 1)”. If prErrorFlag=0 then prErrMessage is the null string.
- prErrFix is used to trigger the running of the prErrFixCmd command. prErrFix must be SNMP SET to 1 to run the command. This can either be achieved with an external SET command or by using the DisMan Event MIB

The advantage of the UCD-SNMP-MIB is that it can count the number of instances of a process and raise an alert if the count is not within configured maximum / minimum limits. It also has the possibility of taking action to rectify a process problem. However, it cannot monitor for process path names or parameters.

Thus, the UCD-SNMP-MIB satisfies requirements 1, 3, 6, 7 and 8 above (monitoring for a process, monitoring the number of instances of a process within maximum / minimum limits, alerting on a process problem, and automatic recovery).

To examine the UCD_SNMP_MIB process information on a target device using SNMP V1 and a community name of public, use:

- `snmpwalk -v 1 -c public zen232 prTable`

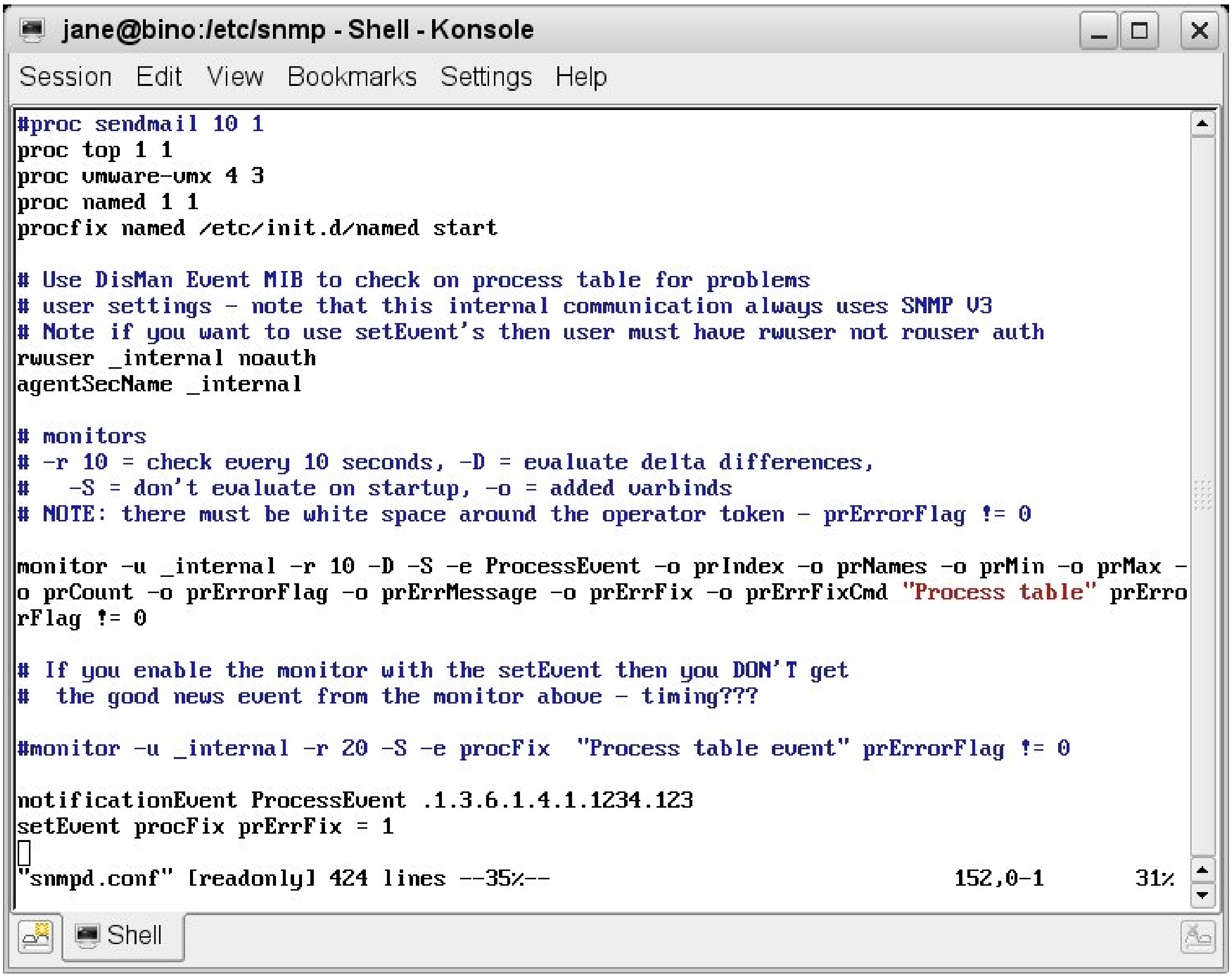
Of course, it is perfectly possible to combine UCD-SNMP-MIB process monitoring with Host Resources MIB process monitoring.

2.3 DisMan Event MIB

The UCD-SNMP-MIB does not automatically raise any TRAPs or NOTIFICATIONs, nor will it run any procfix commands, by default. The DisMan Event MIB, described in RFC 2981, can be used with the prTable to achieve this.

“monitor” configuration lines can be added to snmpd.conf to monitor the value of a MIB OID on the local agent; for process monitoring, the prErrorFlag is the obvious OID to monitor for a value of 1. The monitor configuration can optionally raise a TRAP or NOTIFICATION. monitor can also be used to trigger a change (SNMP SET) in a prErrFix value, thus initiating a recovery script.

monitor configuration lines mandate a username parameter as the local MIB OIDs will be queried (SNMP GET) and, in the case of changing prErrFix, an OID will be changed (SNMP SET). For this internal querying, SNMP V3 is always used, regardless of what version of SNMP is used for external devices to query the local agent. When configuring SNMP V3 users for DisMan Event MIB monitoring, do ensure that the user has read/write access if you need to change the prErrFix MIB value.


jane@bino:/etc/snmp - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
#proc sendmail 10 1
proc top 1 1
proc vmware-vmx 4 3
proc named 1 1
procfix named /etc/init.d/named start

# Use DisMan Event MIB to check on process table for problems
# user settings - note that this internal communication always uses SNMP V3
# Note if you want to use setEvent's then user must have rwuser not rouser auth
rwuser _internal noauth
agentSecName _internal

# monitors
# -r 10 = check every 10 seconds, -D = evaluate delta differences,
#   -S = don't evaluate on startup, -o = added varbinds
# NOTE: there must be white space around the operator token - prErrorFlag != 0

monitor -u _internal -r 10 -D -S -e ProcessEvent -o prIndex -o prNames -o prMin -o prMax -
o prCount -o prErrorFlag -o prErrMessage -o prErrFix -o prErrFixCmd "Process table" prErro
rFlag != 0

# If you enable the monitor with the setEvent then you DON'T get
#  the good news event from the monitor above - timing???

#monitor -u _internal -r 20 -S -e procFix  "Process table event" prErrorFlag != 0

notificationEvent ProcessEvent .1.3.6.1.4.1.1234.123
setEvent procFix prErrFix = 1

"snmpd.conf" [readonly] 424 lines --35%--                        152,0-1       31%
```

Figure 1: snmpd.conf with process and DisMan Event configuration lines

Note when configuring monitor statements for the DisMan Event MIB, there **must** be white space around operators.

In Figure 1 above, four processes are monitored, each having max/min parameters; in addition, named has a procfix line.

A user called *_internal* is created for SNMP V3 use with read/write access; no authentication is required. The monitor statement requires a "-u" parameter which specifies an *agentSecName* – hence the agentSecName definition defining _internal as a valid user for monitor queries.

The uncommented monitor line provides an example that checks each prErrorFlag in the prTable (ie one check for each defined process) for a value !=0. On this condition, the *-e* flag is used to generate an SNMP notification called *ProcessEvent*, which is defined at the bottom of Figure 1. The *-e* parameter can either specify your own TRAP / NOTIFICATION (as shown here) or can use any TRAP / NOTIFICATION that is defined and available to the agent in a MIB file. The event is passed a number of variables (varbinds), each specified with a *-o* parameter (wildcard) and the name of the OID to be sent. For a wildcarded expression, the suffix of the matched instance will be added to any OIDs specified. Thus if *named* is index 3 in the prTable and prErrorFlag.3 is tested and found to be !=0, then the values of prIndex.3, prNames.3, prMin.3 etc. will be included on the event as varbinds. The next-to-last field in the monitor line (*"Process table"* in this case) is an administrative name for this expression, and is used for indexing the mteTriggerTable (and related tables).

The active monitor line checks the prErrorFlag instances every 10 seconds (*-r 10*) and evaluates delta differences (*-D*); the monitor is **not** run on snmpd agent startup (*-S*).

Note that a monitor line only specifies *what* event will be sent and under *which* conditions. A standard snmpd.conf *trapsink* line (or lines) will be necessary to indicate *where* events should be sent to.

The effect of the active monitor line is to send an SNMP notification with enterprise OID .1.3.6.1.4.1.1234.123 including varbinds that report the problem, whenever a process fails to meet its configured criteria. When the problem goes away, an event with the same OID will be sent and the varbinds will indicate the "good news" nature of the event.

The second, commented-out monitor line in Figure 1 demonstrates local automation by running a SET event, *procfix*, when a prErrorFlag instance != 0. The corresponding instance of prErrFix is set to 1 which will trigger any configured procfix action. In the case of a failed *named*, this will cause */etc/init.d/named start* to be run.

On my system, SuSE 10 with net-SNMP-5.4.1-19.4, I found that **either** the bad news / good news events would work, **or** the automatic procfix process restart would work; however if both lines were configured then the "good news" event when the process was healthy again, was never sent. For this reason, the second monitor line is commented out – it is simple enough to configure an action at Zenoss to perform an SNMP SET on the instance of prErrFix to set the value to 1 and cause the procfix action to be executed.

In summary, adding the DisMan Event MIB configuration to an SNMP agent satisfies the initial process management requirements 7 and 8 (alerting on process failure and recovery, and automatic recovery from process failure).

3 Monitoring processes with SSH

There are three ways that SSH can be used to help achieve process monitoring:

- Use SSH to run operating system commands (either built-in (*ps* variations) or scripts)
- Use SSH to run Nagios plugin commands (such as *check_procs*)
- Use SSH to run Zenoss plugins to deliver process information (eg. *zenplugin.py process sshd*)

These options do not inherently rely on having Zenoss as the management system (even the Zenoss plugins operate standalone). This chapter will discuss the basic techniques of SSH, Nagios and Zenoss plugins. Chapter 5 will then discuss how these SSH methods can be incorporated with a Zenoss management system.

Nagios plugins offer the advantage of a large library of system and network management checks that are coded to a defined format. Zenoss understands the output of Nagios plugins and can use it automatically to generate events.

The disadvantage of using Nagios plugins with Zenoss is that you have to install the Nagios plugins on any targets that you want to access that way – you have the old problem of installing and maintaining an "agent".

Similarly, the Zenoss plugins provide some pre-coded functionality but they have to be installed along with Python. Zenoss has several performance data collection templates that use Zenoss plugins – look under the *Templates* tab for */Devices/Server/Cmd* at the *Devices*, *FileSystem* and *ethernetCsmacd* templates.

A compromise might be to write native scripts that produce output in Nagios format which removes the need to install an "agent" remotely (though you still have to get the script delivered to the targets).

3.1 Setting up SSH

Most Unix / Linux Operating Systems come with an SSH implementation. PuTTY is probably the best known SSH for Windows platforms. Communication is protected by encryption which usually requires public/private key pairs to be generated. The private key needs to be held on the SSH client (for example, a Zenoss manager); the public key is needed on the SSH server (for example, a device running sshd).

Typically on a Unix / Linux system, any user that runs SSH will have a *.ssh* directory under their home directory which contains SSH key files; it should have 600 access permissions.

The key pairs are generated with a utility generally called *ssh-keygen*. SSH can use either RSA or DSA as an authentication algorithm and there are 2 versions of the SSH protocol – version 1 and version 2. Most modern implementations of SSH should be using the DSA algorithm and SSH version 2. So, if you want to use SSH with a Zenoss management system, using the userid of *zenoss*, to manage a remote system called *bino* with a userid of *zenrem*, generate a public/private key pair using DSA, for SSH version 2, by:

- Becoming the *zenoss* user on the management system (because of the way this user is created, you may need to su to *root* and then run *su - zenoss*)
- *ssh-keygen -t dsa* you will be prompted for a passphrase which may be blank
- inspect *~/.ssh* for i*d_dsa* and *id_dsa.pub* and check the directory has 600 access permissions

- copy *id_dsa.pub* to the machine *bino* into the *.ssh* subdirectory of the userid *zenrem*. It should be copied into the file *authorized_keys (*or appended to *authorized_keys* if the file already exists).
- The private key, *id_dsa ,*remains on the Zenoss system. It must have 600 access permissions.
- The public key can be copied to the *authorized_keys* file of as many systems as you want to manage.

Note that some implementation of SSH use a filename *authorized_keys**2*** to hold version 2 DSA public keys.

If you specify a passphrase when generating the key pairs, this passphrase is used to further protect access to the private key, *id_dsa* and you will be prompted for the passphrase before any SSH communication can take place.

Note that the names *id_dsa* and *id_dsa.pub* are defaults. It is perfectly possible to use different file names and then to specify the keyfile name as part of the ssh command.

So, if we have a user, *zenrem* on a managed system, *bino*, with the correct public key in *zenrem*'s *.ssh/authorized_keys* file, you can test the communication from the Zenoss system, as user *zenoss*, with:

- *ssh zenrem@bino*
- If you have a passphrase configured, you will be prompted for it (this prompt is from the **local** Zenoss system to access the **local** private key).
- If this is the first SSH communication with *bino*, an RSA key for the host *bino* will be generated and you will be asked whether to continue connection. If you answer *Yes* then this host key will be added into the file *known_hosts* under *zenoss*'s .ssh directory.

In general, key pairs may be used symmetrically; that is, if both client and server have the same *id_dsa* private key and the same matching public key in their *authorized_keys* file, then either can act as client (*ssh* command) or server (*sshd* daemon).

Note that testing SSH with a user *zenoss* on the server side (ie ssh'ing **in** to a Zenoss management system) will not work as the standard Zenoss install does not permit logins to the user called *zenoss* – this also inhibits SSH access (note that a root user **can** use *su – zenoss* to become the zenoss user but this does not help the SSH scenario).

In summary, you need the private key, *id_dsa*, to authorize communication **out** of your system (ie. acting as an SSH client); you need the public key in the file *authorized_keys* to authorize communication **in** (ie acting as an SSH server). You don't actually need the public key in the file *id_dsa.pub*.

3.1.1 Using to SSH to directly monitor processes

Once SSH communications is correctly established, **any** script can be run on a remote system, hence any requirements for process monitoring could be met; whether monitoring for a single process instance, multiple instances, exact process names with or without process parameters. It is also possible to code recovery actions and to generate alerts – SNMP TRAPs, messages to syslog, emails, or any other form of notification. The negative aspect of direct SSH communication is that, if a script is run, then the script (and any embedded scripts or calls) somehow have to be distributed to the target.

3.2 Nagios plugin architecture

The Zenoss Developer's Guide (page 18 of the 2.3 version) provides a reference to Nagios plugin API documentation at

http://nagiosplug.sourceforge.net/developer-guidelines.html#PLUGOUTPUT

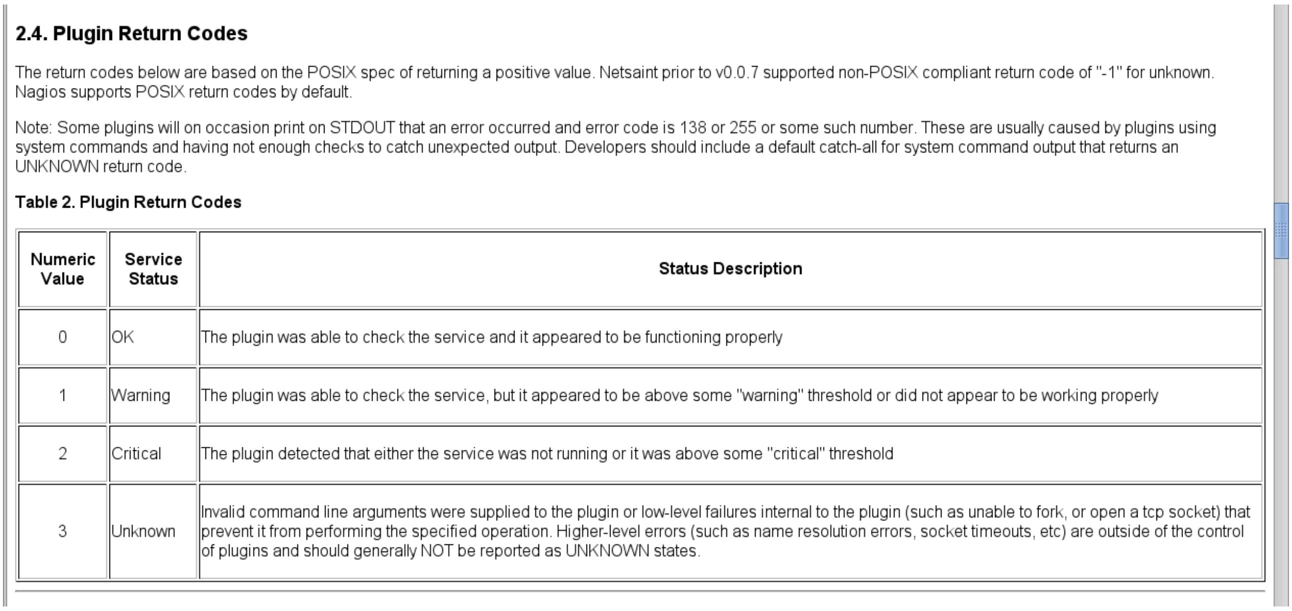
Chapter 2 of this Nagios paper documents the output format for:

- status result of the plugin
- any performance data delivered by the plugin

Basically, Nagios should deliver **one** line of output. Status output should be in the format:

SERVICE STATUS: Information text

Valid return codes are documented as shown in the figure below.



2.4. Plugin Return Codes

The return codes below are based on the POSIX spec of returning a positive value. Netsaint prior to v0.0.7 supported non-POSIX compliant return code of "-1" for unknown. Nagios supports POSIX return codes by default.

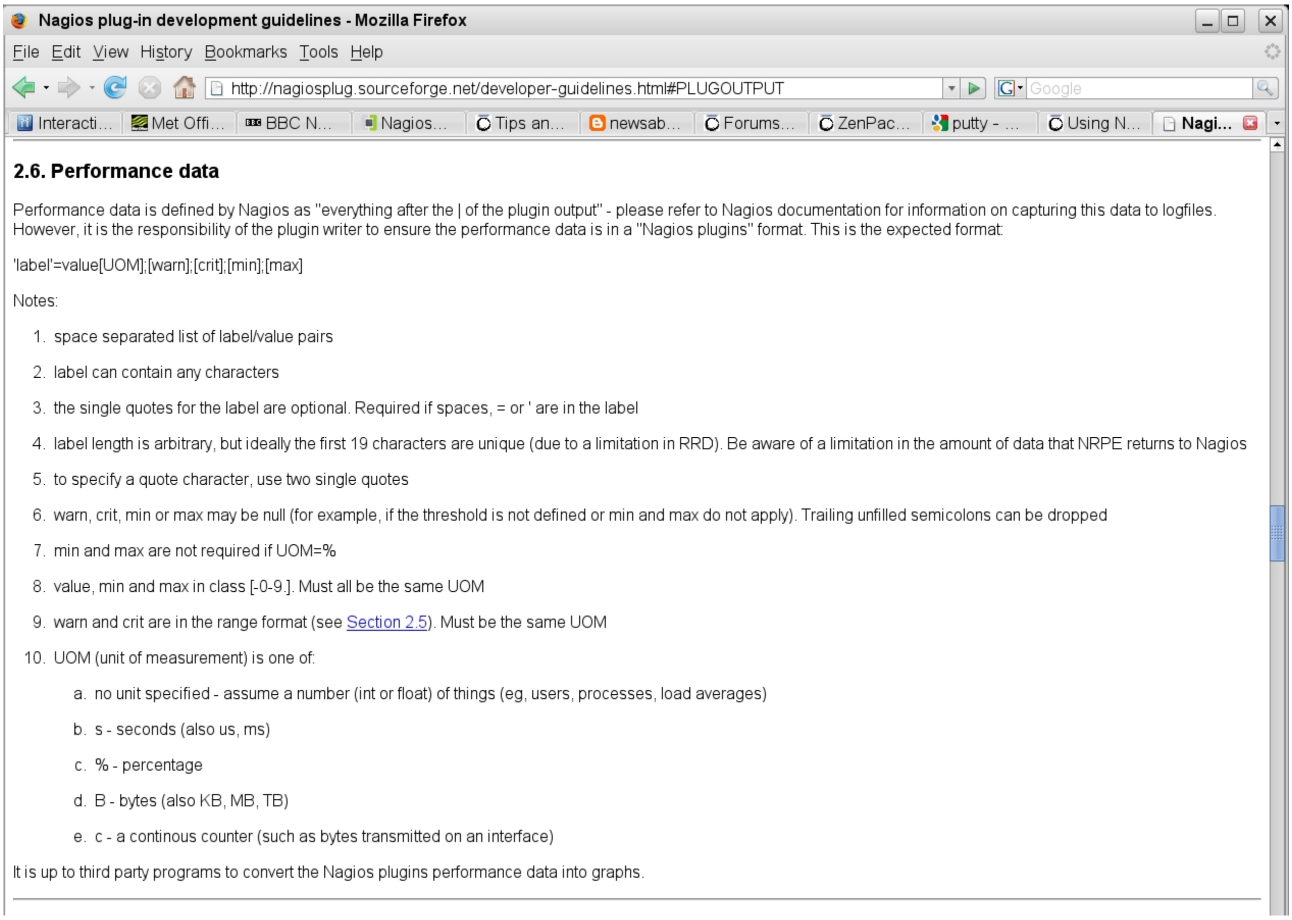
Note: Some plugins will on occasion print on STDOUT that an error occurred and error code is 138 or 255 or some such number. These are usually caused by plugins using system commands and having not enough checks to catch unexpected output. Developers should include a default catch-all for system command output that returns an UNKNOWN return code.

Table 2. Plugin Return Codes

Numeric Value	Service Status	Status Description
0	OK	The plugin was able to check the service and it appeared to be functioning properly
1	Warning	The plugin was able to check the service, but it appeared to be above some "warning" threshold or did not appear to be working properly
2	Critical	The plugin detected that either the service was not running or it was above some "critical" threshold
3	Unknown	Invalid command line arguments were supplied to the plugin or low-level failures internal to the plugin (such as unable to fork, or open a tcp socket) that prevent it from performing the specified operation. Higher-level errors (such as name resolution errors, socket timeouts, etc) are outside of the control of plugins and should generally NOT be reported as UNKNOWN states.

Figure 2: Nagios plugin return codes

If the plugin delivers performance data, it must follow the return code and text, separated from it by the vertical bar symbol.


Nagios plug-in development guidelines - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://nagiosplug.sourceforge.net/developer-guidelines.html#PLUGOUTPUT
Google
Interacti... Met Offi... BBC N... Nagios... Tips an... newsab... Forums... ZenPac... putty - ... Using N... Nagi...

2.6. Performance data

Performance data is defined by Nagios as "everything after the | of the plugin output" - please refer to Nagios documentation for information on capturing this data to logfiles. However, it is the responsibility of the plugin writer to ensure the performance data is in a "Nagios plugins" format. This is the expected format:

'label'=value[UOM];[warn];[crit];[min];[max]

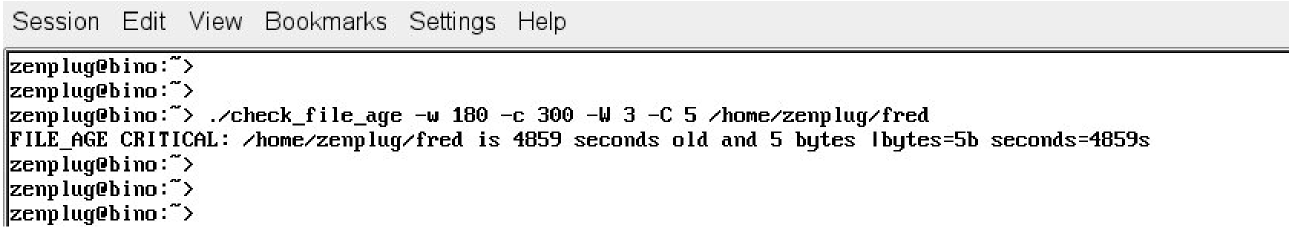
Notes:

1. space separated list of label/value pairs
2. label can contain any characters
3. the single quotes for the label are optional. Required if spaces, = or ' are in the label
4. label length is arbitrary, but ideally the first 19 characters are unique (due to a limitation in RRD). Be aware of a limitation in the amount of data that NRPE returns to Nagios
5. to specify a quote character, use two single quotes
6. warn, crit, min or max may be null (for example, if the threshold is not defined or min and max do not apply). Trailing unfilled semicolons can be dropped
7. min and max are not required if UOM=%
8. value, min and max in class [-0-9.]. Must all be the same UOM
9. warn and crit are in the range format (see Section 2.5). Must be the same UOM
10. UOM (unit of measurement) is one of:
    a. no unit specified - assume a number (int or float) of things (eg, users, processes, load averages)
    b. s - seconds (also us, ms)
    c. % - percentage
    d. B - bytes (also KB, MB, TB)
    e. c - a continous counter (such as bytes transmitted on an interface)

It is up to third party programs to convert the Nagios plugins performance data into graphs.

Figure 3: Nagios plugin format for delivering performance data

As an example, the check_file_age Nagios plugin takes warning and critical parameters for age (-w and -c parameters in seconds) and size (-W and -C parameters in bytes). To get the usage for any Nagios plugin, use the *-h* parameter after the plugin command name (*check_file_age -h*). Thus a Nagios plugin, check_file_age, might respond as shown below:

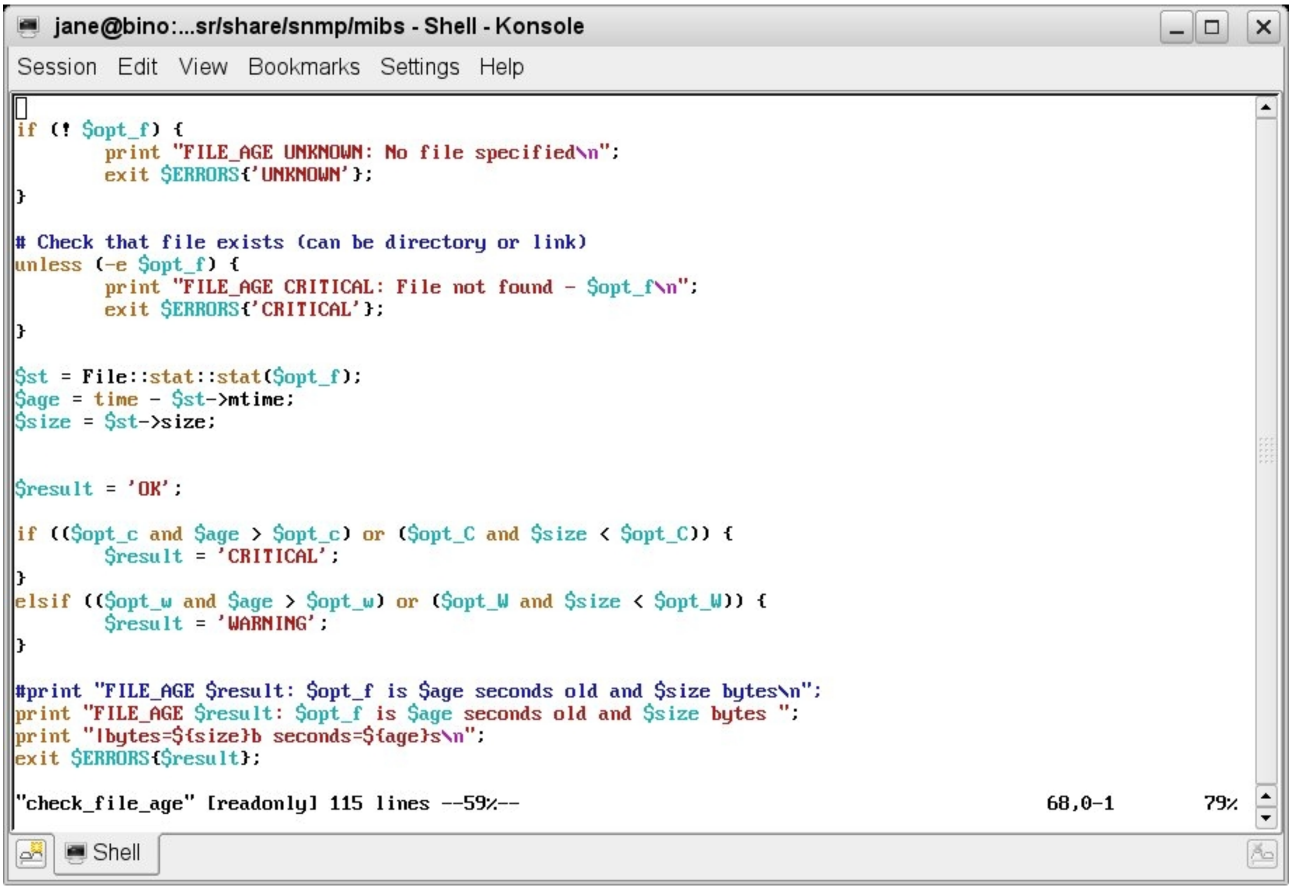


```
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> ./check_file_age -w 180 -c 300 -W 3 -C 5 /home/zenplug/fred
FILE_AGE CRITICAL: /home/zenplug/fred is 4859 seconds old and 5 bytes |bytes=5b seconds=4859s
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
```

Figure 4: Nagios plugin check_file_age with performance output

Note that this plugin has been modified from the standard Nagios plugin in order to deliver performance data after the vertical bar.

The plugin is actually a Perl script, the main body of which is shown below:


jane@bino:...sr/share/snmp/mibs - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
if (! $opt_f) {
        print "FILE_AGE UNKNOWN: No file specified\n";
        exit $ERRORS{'UNKNOWN'};
}

# Check that file exists (can be directory or link)
unless (-e $opt_f) {
        print "FILE_AGE CRITICAL: File not found - $opt_f\n";
        exit $ERRORS{'CRITICAL'};
}

$st = File::stat::stat($opt_f);
$age = time - $st->mtime;
$size = $st->size;


$result = 'OK';

if (($opt_c and $age > $opt_c) or ($opt_C and $size < $opt_C)) {
        $result = 'CRITICAL';
}
elsif (($opt_w and $age > $opt_w) or ($opt_W and $size < $opt_W)) {
        $result = 'WARNING';
}

#print "FILE_AGE $result: $opt_f is $age seconds old and $size bytes\n";
print "FILE_AGE $result: $opt_f is $age seconds old and $size bytes ";
print "|bytes=${size}b seconds=${age}s\n";
exit $ERRORS{$result};

"check_file_age" [readonly] 115 lines --59%--                     68,0-1        79%
```

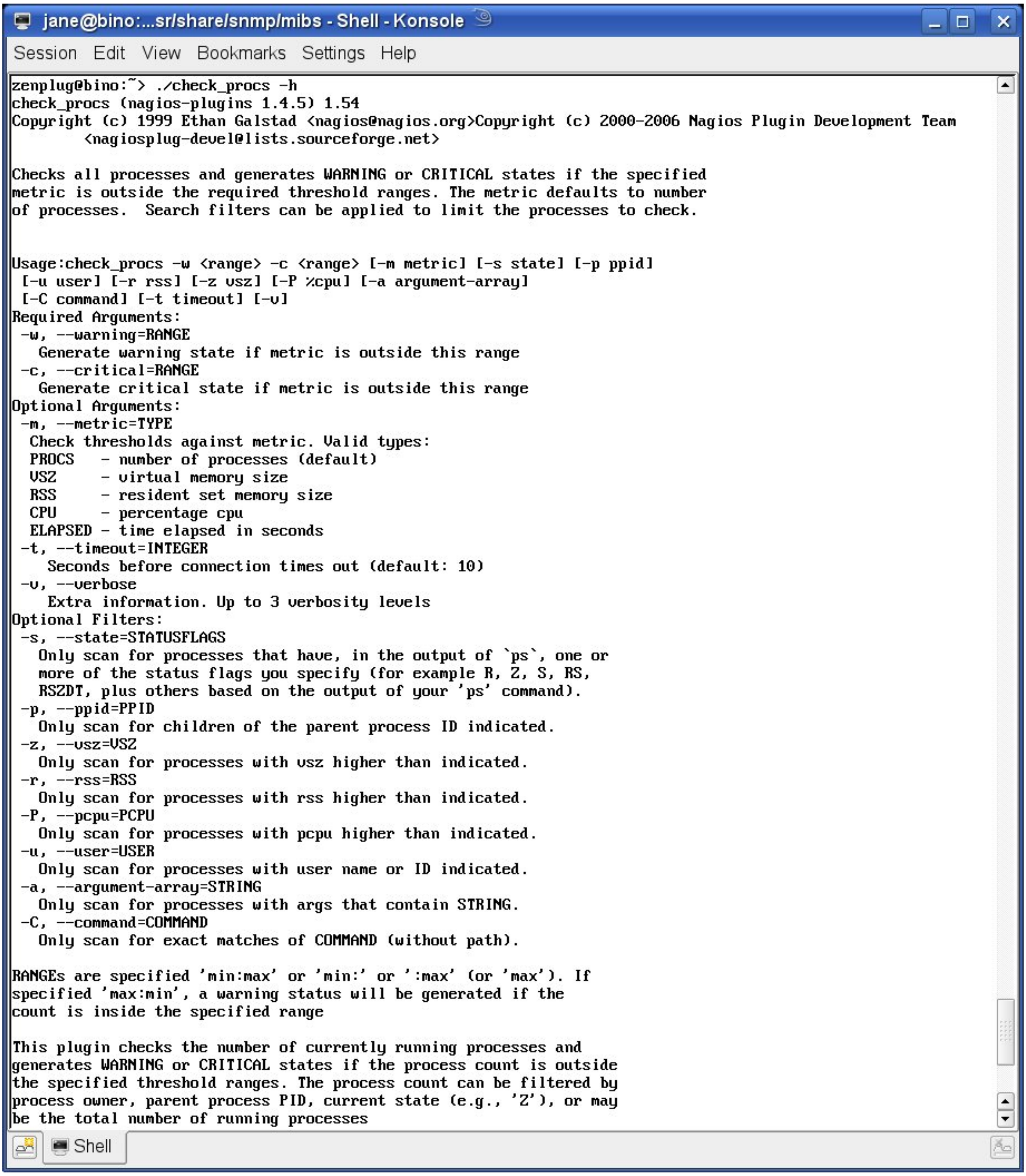
Figure 5: Body of modified check_file_age Nagios plugin

The first two sections check that a file name has been supplied and that a supplied file name exists, each returning an output line with a result code (UNKNOWN or CRITICAL). Note that an exit status is supplied as well as the status as part of the output line. The main body of the script checks the file age and size against warning and critical thresholds. The end of the script then delivers the output line with the result code, information text and the values for size and age; again the exit status is delivered.

Nagios plugins are installed as standard on a Zenoss server under */usr/local/zenoss/common/libexec* . Nagios plugins can also be installed on remote systems and run standalone. Note that many require the *utils.pm* file to be available either in the same directory as the plugin or in an include path, @INC. If you receive an error message saying that *utils.pm* cannot be located, check the reported @INC path and a symbolic link can be provided from the actual *utils.pm* directory to one of the directories in the path.

3.2.1 Using Nagios plugins to monitor processes

The standard Nagios plugins include a *check_procs* plugin which can be installed standalone on a device.

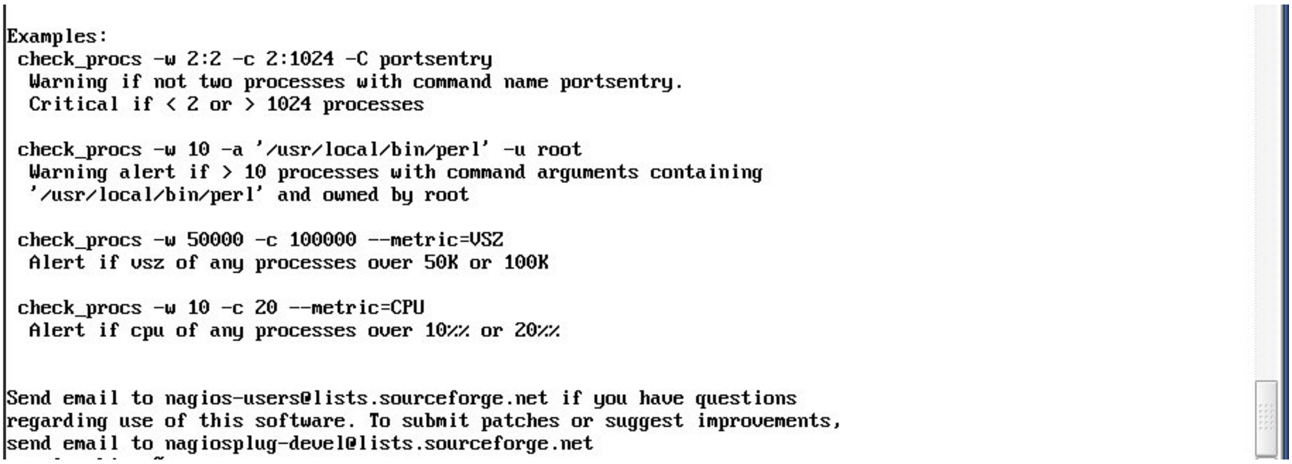


jane@bino:...sr/share/snmp/mibs - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
zenplug@bino:~> ./check_procs -h
check_procs (nagios-plugins 1.4.5) 1.54
Copyright (c) 1999 Ethan Galstad <nagios@nagios.org>Copyright (c) 2000-2006 Nagios Plugin Development Team
        <nagiosplug-devel@lists.sourceforge.net>

Checks all processes and generates WARNING or CRITICAL states if the specified
metric is outside the required threshold ranges. The metric defaults to number
of processes.  Search filters can be applied to limit the processes to check.


Usage:check_procs -w <range> -c <range> [-m metric] [-s state] [-p ppid]
 [-u user] [-r rss] [-z vsz] [-P %cpu] [-a argument-array]
 [-C command] [-t timeout] [-v]
Required Arguments:
 -w, --warning=RANGE
   Generate warning state if metric is outside this range
 -c, --critical=RANGE
   Generate critical state if metric is outside this range
Optional Arguments:
 -m, --metric=TYPE
  Check thresholds against metric. Valid types:
  PROCS   - number of processes (default)
  VSZ     - virtual memory size
  RSS     - resident set memory size
  CPU     - percentage cpu
  ELAPSED - time elapsed in seconds
 -t, --timeout=INTEGER
    Seconds before connection times out (default: 10)
 -v, --verbose
    Extra information. Up to 3 verbosity levels
Optional Filters:
 -s, --state=STATUSFLAGS
   Only scan for processes that have, in the output of `ps`, one or
   more of the status flags you specify (for example R, Z, S, RS,
   RSZDT, plus others based on the output of your 'ps' command).
 -p, --ppid=PPID
   Only scan for children of the parent process ID indicated.
 -z, --vsz=VSZ
   Only scan for processes with vsz higher than indicated.
 -r, --rss=RSS
   Only scan for processes with rss higher than indicated.
 -P, --pcpu=PCPU
   Only scan for processes with pcpu higher than indicated.
 -u, --user=USER
   Only scan for processes with user name or ID indicated.
 -a, --argument-array=STRING
   Only scan for processes with args that contain STRING.
 -C, --command=COMMAND
   Only scan for exact matches of COMMAND (without path).

RANGEs are specified 'min:max' or 'min:' or ':max' (or 'max'). If
specified 'max:min', a warning status will be generated if the
count is inside the specified range

This plugin checks the number of currently running processes and
generates WARNING or CRITICAL states if the process count is outside
the specified threshold ranges. The process count can be filtered by
process owner, parent process PID, current state (e.g., 'Z'), or may
be the total number of running processes
```

Figure 6: Help for the check_procs Nagios plugin

Examples are also given at the end of the help:



```
Examples:
 check_procs -w 2:2 -c 2:1024 -C portsentry
  Warning if not two processes with command name portsentry.
  Critical if < 2 or > 1024 processes

 check_procs -w 10 -a '/usr/local/bin/perl' -u root
  Warning alert if > 10 processes with command arguments containing
  '/usr/local/bin/perl' and owned by root

 check_procs -w 50000 -c 100000 --metric=VSZ
  Alert if vsz of any processes over 50K or 100K

 check_procs -w 10 -c 20 --metric=CPU
  Alert if cpu of any processes over 10%% or 20%%


Send email to nagios-users@lists.sourceforge.net if you have questions
regarding use of this software. To submit patches or suggest improvements,
send email to nagiosplug-devel@lists.sourceforge.net
```

Figure 7: Examples for using Nagios check_procs plugin

Note that extra output can be achieved with the *-vvv* option (LOTS of verbosity). In the case of the check_procs plugin, this extra flag shows that the command that is actually run is:

```
/bin/ps axwo 'stat uid pid ppid vsz rss pcpu comm args'
```

When considering the process management requirements at the beginning of this document, the Nagios plugins have possibilities for addressing 1, 3, 5 and 6 (monitoring single and multiple instances of processes by short process name and by considering the parameters of a process). There is no ability in the standard plugin to take remedial action or to send alerts; however, the Nagios API is just that and it is perfectly possible to write your own plugin or to modify some of the standard plugins provided. In addition, the Nagios plugin allows monitoring based on resources used, such as memory and CPU, although no performance data values are returned by the default plugin.

3.3 Zenoss plugins

Zenoss plugins are entirely separate from Nagios plugins. They are also sometimes referred to as ZenPlugins (or even just “plugins”) in the documentation. They are a collection of platform-specific Python libraries and the zenplugin.py command. They can be used to collect information using SSH, from remote systems. The Zenoss plugins are only useful to monitor a remote system if that system has Python installed and if the Zenoss plugins are supported on the architecture. Zenoss ships Linux, FreeBSD and Darwin; Solaris and OpenBSD variants are available in the community. **Note** that although Zenoss itself requires Python 2.4 to be installed, the Zenoss plugins work with both Python 2.4 and 2.5.

Note that the Zenoss plugins are **only** used for collecting performance data; they are not a pre-requisite for **modelling** a device.

The Zenoss plugins can be downloaded from the Zenoss download site (http://www.zenoss.com/download/links?creg=no) under the heading “Remote Monitoring Scripts” . Good overview information is available at the end of the Zenoss FAQ at http://www.zenoss.com/community/docs/faqs/faq-english/ . There is also a Zenoss plugins HowTo at http://www.zenoss.com/community/docs/howtos/zenoss-plugins . I found the documentation for installing the Zenoss plugins rather confusing; the following process worked successfully on both SLES 10 (32 bit) and Open SuSE 10.2 (64 bit).

Note that both python and the python **development** package must be already installed. **Note also** that you need to install the Python setuptools package or you are likely to get an error message about an ApplicationError - “ImportError: No module named common”.

I found the easiest way to install the Zenoss plugins was to:

1. Get the latest Zenoss plugins package from http://www.zenoss.com/download/links?creg=no . I used the “Other” source tarball under the “Remote Monitoring Scripts” section and got Zenoss-Plugins-2.0.4.tar.gz
2. Get the source tarball for the Python setuptools utility from http://pypi.python.org/packages/source/s/setuptools/ (I got setuptools-0.6c9.tar.gz)
3. As root, untar the Zenoss plugins file
4. Change to the Zenoss-Plugins-2.0.4 directory
5. Run

```
python ./setup.py build
python ./setup.py install
```

6. Python packages typically get installed to */usr/local/lib/python2.5/site-packages* (the directory will be created if necessary)
7. Untar the setuptools file
8. Change to the setuptools-0.6c9 directory
9. Run

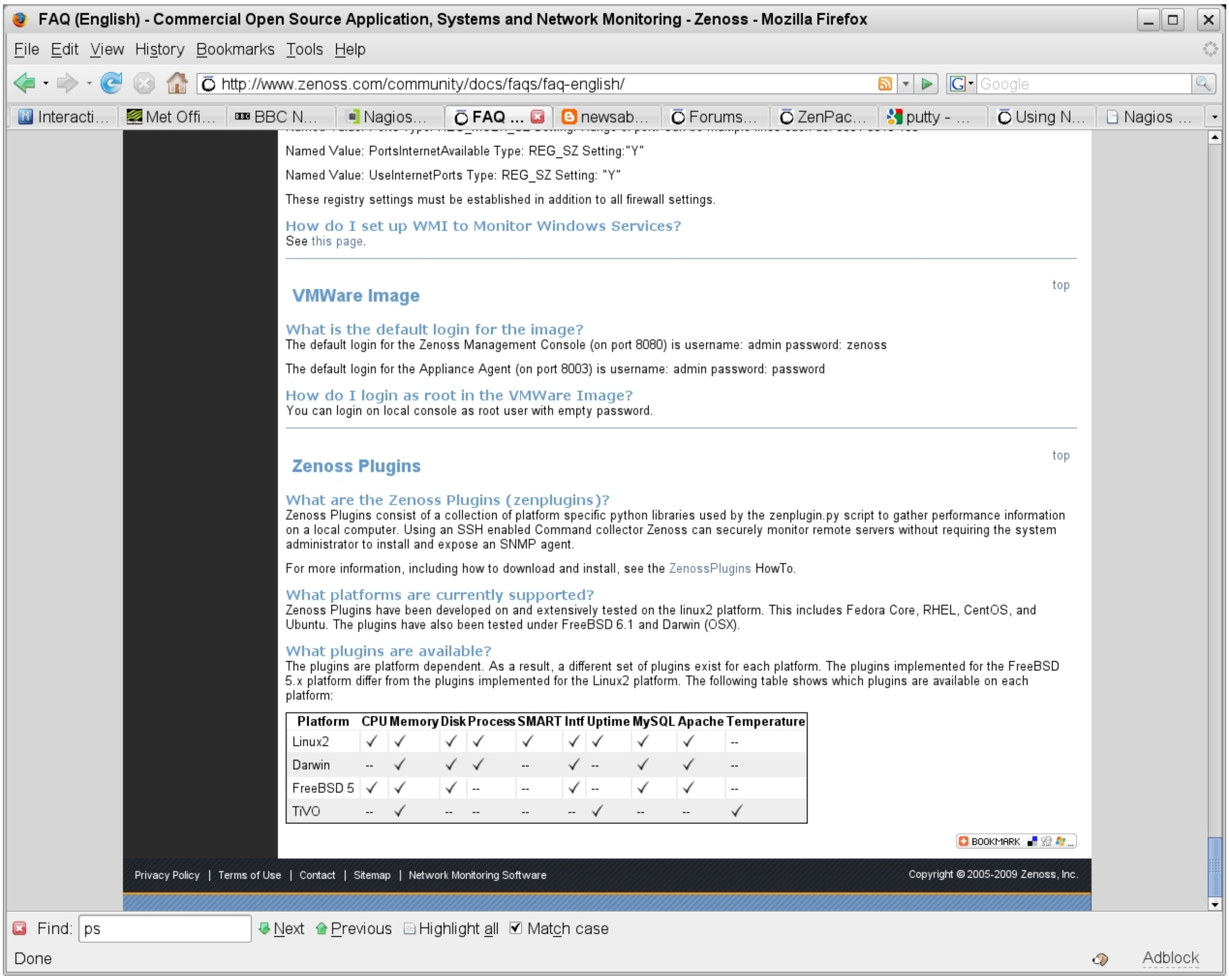
```
python ./setup.py install
```

10. As a normal user, test with

```
zenplugin.py --list-plugins
```

11. Note that zenplugin.py will be installed into */usr/local/bin*

The FAQ documents what utilities are supported on which architecture:


FAQ (English) - Commercial Open Source Application, Systems and Network Monitoring - Zenoss - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.zenoss.com/community/docs/faqs/faq-english/
Google
Interacti… Met Offi… BBC N… Nagios… FAQ … newsab… Forums… ZenPac… putty - … Using N… Nagios …

Named Value: PortsInternetAvailable Type: REG_SZ Setting:"Y"

Named Value: UseInternetPorts Type: REG_SZ Setting: "Y"

These registry settings must be established in addition to all firewall settings.

How do I set up WMI to Monitor Windows Services?
See this page.

top

VMWare Image

What is the default login for the image?
The default login for the Zenoss Management Console (on port 8080) is username: admin password: zenoss

The default login for the Appliance Agent (on port 8003) is username: admin password: password

How do I login as root in the VMWare Image?
You can login on local console as root user with empty password.

top

Zenoss Plugins

What are the Zenoss Plugins (zenplugins)?
Zenoss Plugins consist of a collection of platform specific python libraries used by the zenplugin.py script to gather performance information on a local computer. Using an SSH enabled Command collector Zenoss can securely monitor remote servers without requiring the system administrator to install and expose an SNMP agent.

For more information, including how to download and install, see the ZenossPlugins HowTo.

What platforms are currently supported?
Zenoss Plugins have been developed on and extensively tested on the linux2 platform. This includes Fedora Core, RHEL, CentOS, and Ubuntu. The plugins have also been tested under FreeBSD 6.1 and Darwin (OSX).

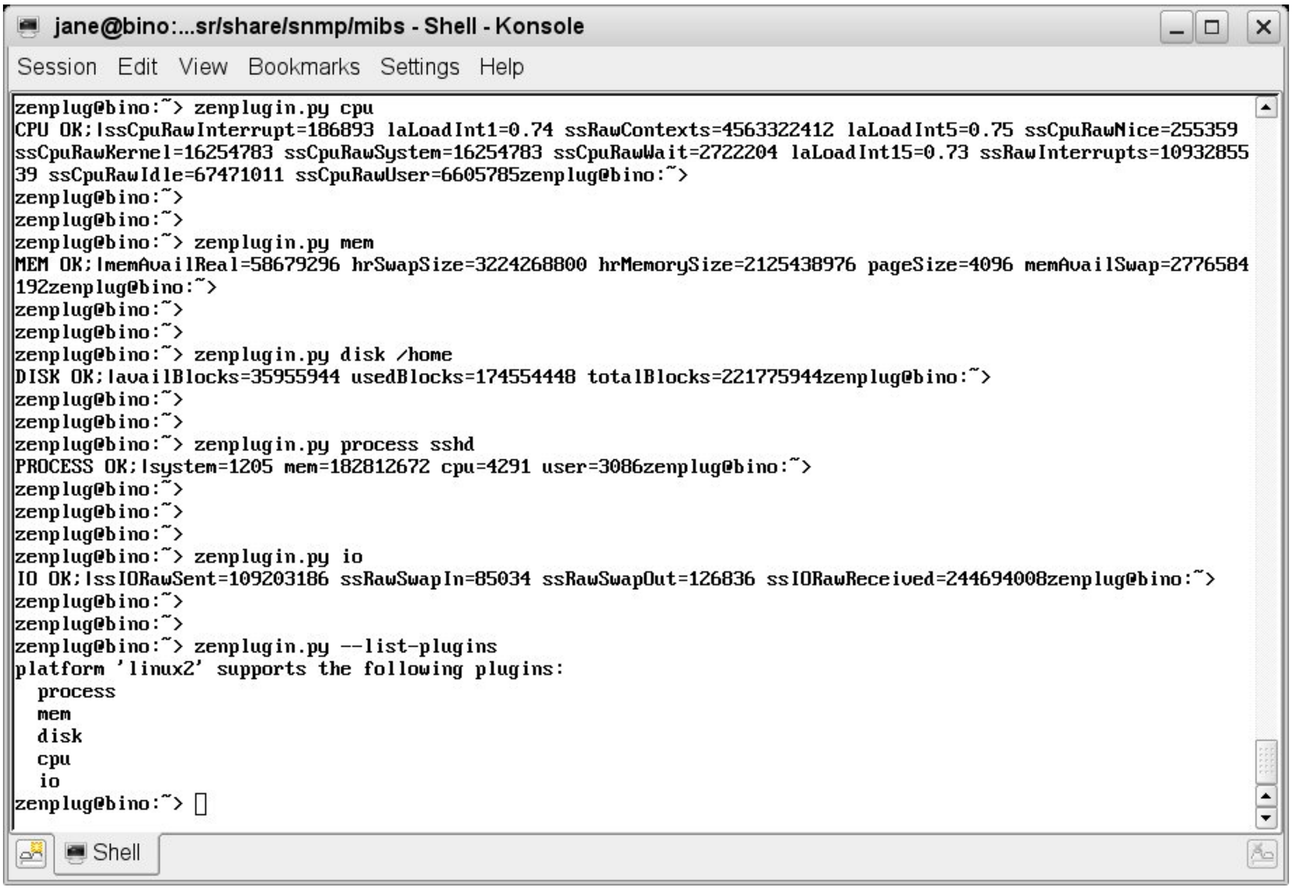
What plugins are available?
The plugins are platform dependent. As a result, a different set of plugins exist for each platform. The plugins implemented for the FreeBSD 5.x platform differ from the plugins implemented for the Linux2 platform. The following table shows which plugins are available on each platform:

Platform	CPU	Memory	Disk	Process	SMART	Intf	Uptime	MySQL	Apache	Temperature
Linux2	✓	✓	✓	✓	✓	✓	✓	✓	✓	--
Darwin	--	✓	✓	✓	--	✓	--	✓	✓	--
FreeBSD 5	✓	✓	✓	--	--	✓	--	✓	✓	--
TiVO	--	✓	--	--	--	--	✓	--	--	✓

BOOKMARK
Privacy Policy | Terms of Use | Contact | Sitemap | Network Monitoring Software
Copyright © 2005-2009 Zenoss, Inc.
Find: ps
Next
Previous
Highlight all
☑ Match case
Done
Adblock

Figure 8: Zenoss FAQ for Zenoss plugins

I repeat - **Zenoss plugins are entirely separate from Nagios plugins**. However, the Zenoss plugins implement the output specification of Nagios commands. Note in the examples shown in Figure 9 that the return code is printed along with informational text, followed by a vertical bar, followed by one or more performance data values. Various Zenoss performance data collector templates, under */Server/Cmd/Linux*, use the Zenoss plugins to deliver data values for graphs for *Devices*, *FileSystem* and *ethernetCsmacd* templates.



```
zenplug@bino:~> zenplugin.py cpu
CPU OK;|ssCpuRawInterrupt=186893 laLoadInt1=0.74 ssRawContexts=4563322412 laLoadInt5=0.75 ssCpuRawNice=255359
ssCpuRawKernel=16254783 ssCpuRawSystem=16254783 ssCpuRawWait=2722204 laLoadInt15=0.73 ssRawInterrupts=10932855
39 ssCpuRawIdle=67471011 ssCpuRawUser=6605785zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> zenplugin.py mem
MEM OK;|memAvailReal=58679296 hrSwapSize=3224268800 hrMemorySize=2125438976 pageSize=4096 memAvailSwap=2776584
192zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> zenplugin.py disk /home
DISK OK;|availBlocks=35955944 usedBlocks=174554448 totalBlocks=221775944zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> zenplugin.py process sshd
PROCESS OK;|system=1205 mem=182812672 cpu=4291 user=3086zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> zenplugin.py io
IO OK;|ssIORawSent=109203186 ssRawSwapIn=85034 ssRawSwapOut=126836 ssIORawReceived=244694008zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> zenplugin.py --list-plugins
platform 'linux2' supports the following plugins:
  process
  mem
  disk
  cpu
  io
zenplug@bino:~> 
```

Figure 9: Output from Zenoss plugin commands

3.3.1 Using Zenoss plugins to monitor processes

As can be seen from the screenshot above in Figure 9, there is a process Zenoss plugin that takes a process name as argument. It delivers whether at least one instance of the process is running but does not obviously distinguish between process name and arguments, nor does it help as to the number of instances that are running. There is no concept of the Zenoss plugins running automatic recovery actions or sending alerts (which is reasonable – they are designed as a tool to work with a Zenoss manager which **can** interpret output from the Zenoss plugins and **can** deliver recovery and alerting actions).

4 Monitoring processes with Zenoss's zenprocess daemon

Zenoss has several techniques for managing processes. Fundamentally, there are three separate elements:

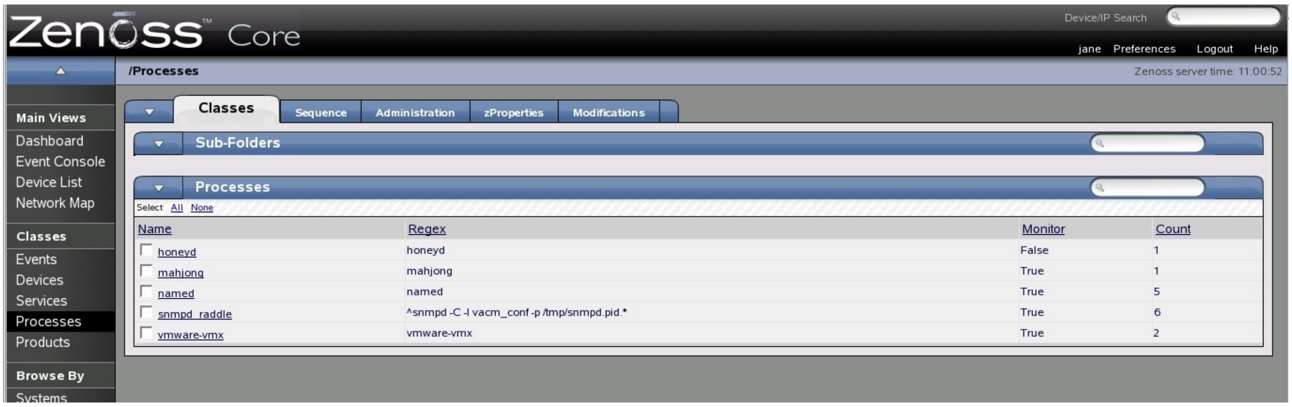
- Process configuration
- Process discovery through the zenmodeler daemon (every 12 hours by default)

- Process status checking through the zenprocess daemon (every 3 minutes by default)

These default polling intervals are controlled from the left-hand *Collectors -> localhost* menu.

4.1 Process configuration

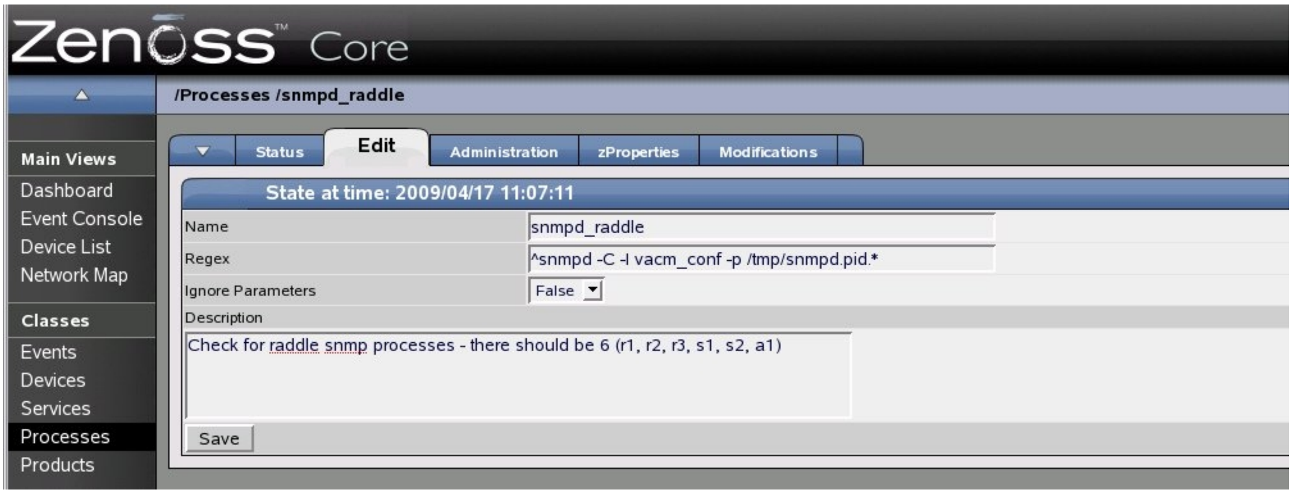
The left-hand menu of the main Zenoss GUI provides a *Processes* menu for configuring processes to monitor. None are configured out-of-the-box.



Name	Regex	Monitor	Count
□ honeyd	honeyd	False	1
□ mahjong	mahjong	True	1
□ named	named	True	5
□ snmpd_raddle	^snmpd -C -l vacm_conf -p /tmp/snmpd.pid.*	True	6
□ vmware-vmx	vmware-vmx	True	2

Figure 10: Zenoss Processes menu

Various parameters are configurable for each process to be monitored:



State at time: 2009/04/17 11:07:11

Name	snmpd_raddle
Regex	^snmpd -C -l vacm_conf -p /tmp/snmpd.pid.*
Ignore Parameters	False

Description

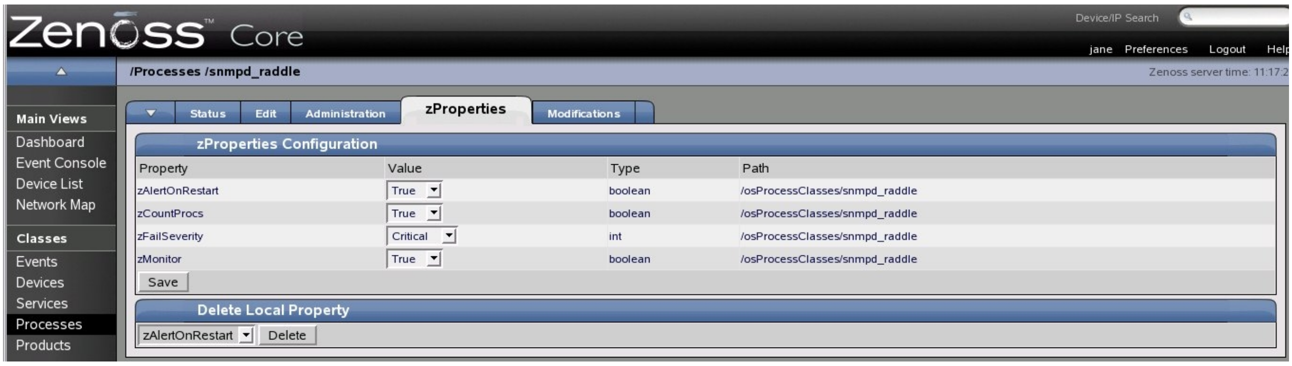
Check for raddle snmp processes - there should be 6 (r1, r2, r3, s1, s2, a1)

Figure 11: Process details that can be edited

The *Name* field is simply a descriptive name – typically reflecting the process name. The Regex field controls what process is monitored. A trivial example, such as in Figure 10 above, shows a regex of *named* which will match any process name that includes *named* and parameters to the process name are ignored. The example in Figure 11 is more specific – the process name must start with snmpd (the ^ specifies start-of-line) and the parameters to the process are also considered when deciding on whether to monitor the process. The regex must match exactly upto the */tmp/snmpd.pid* and can then have any combination of characters following (the *).

Note that with Zenoss 2.3.3 and earlier versions, the *Ignore Parameters* flag sometimes appears to be ignored! For example, in Figure 10 above where *Ignore Parameters* is set to *True* for the named process, processes are automatically detected that have the string "named" in the parameters of **other** commands.

Processes also have zProperties which can further modify behaviour.



/Processes /snmpd_raddle

zProperties Configuration

Property	Value	Type	Path
zAlertOnRestart	True	boolean	/osProcessClasses/snmpd_raddle
zCountProcs	True	boolean	/osProcessClasses/snmpd_raddle
zFailSeverity	Critical	int	/osProcessClasses/snmpd_raddle
zMonitor	True	boolean	/osProcessClasses/snmpd_raddle

Save

Delete Local Property

zAlertOnRestart Delete

Figure 12: zProperties options for a Process

The zProperties are:

● zAlertOnRestart	- generate an event when the process is detected again
● zCountProcs	- it is unclear what effect this has
● zFailSeverity	- the severity of the event generated when the process fails
● zMonitor	- whether to monitor for this process on all devices

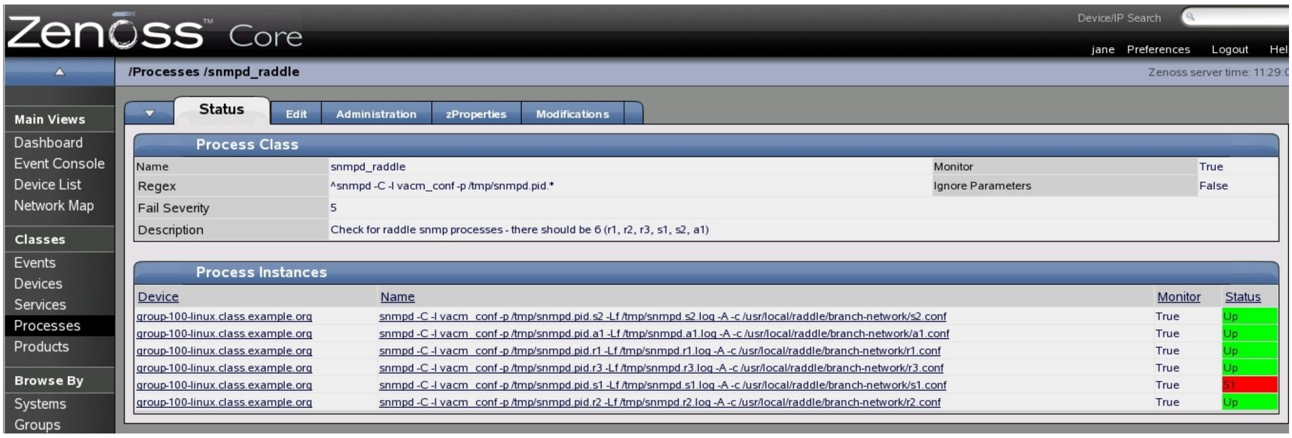
Some of these zProperties are rather problematical. The two associated with events work well. If zAlertOnRestart is set to True, then recovery of a process will result in a "good news" event with a Cleared severity, which will automatically clear a preceding "bad news" event for that process from the same device – this is standard Zenoss event correlation.

The zCountProcs zProperty does not appear to have any effect. There is no opportunity to specify what count is the "correct" number or range. Even if zCountProcs is set to False, data appears to be collected for the number of instances of a process – this can be seen in the performance graphs for a process for a device.

The zMonitor zProperty should specify globally whether to monitor for a process on all discovered devices. For some processes, this would be better set to False and the process monitor can then be activated at the specific device level; however, doing so seems to result in very variable monitoring results (with Zenoss 2.3.3). Process monitoring seems much more reliable with zMonitor set to True.

Although with Zenoss 2.3.3, process configuration appears more stable than with previous versions, there was sometimes a need to restart the zenprocess daemon after process configuration takes place.

The Status tab of a specific process shows how many instances of a process are running, where they are running, and their status:



/Processes /snmpd_raddle

Process Class

Name	snmpd_raddle	Monitor	True
Regex	^snmpd -C -l vacm_conf -p /tmp/snmpd.pid.*	Ignore Parameters	False
Fail Severity	5		
Description	Check for raddle snmp processes - there should be 6 (r1, r2, r3, s1, s2, a1)		

Process Instances

Device	Name	Monitor	Status
group-100-linux.class.example.org	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.s2 -Lf /tmp/snmpd.s2.log -A -c /usr/local/raddle/branch-network/s2.conf	True	Up
group-100-linux.class.example.org	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.a1 -Lf /tmp/snmpd.a1.log -A -c /usr/local/raddle/branch-network/a1.conf	True	Up
group-100-linux.class.example.org	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.r1 -Lf /tmp/snmpd.r1.log -A -c /usr/local/raddle/branch-network/r1.conf	True	Up
group-100-linux.class.example.org	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.r3 -Lf /tmp/snmpd.r3.log -A -c /usr/local/raddle/branch-network/r3.conf	True	Up
group-100-linux.class.example.org	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.s1 -Lf /tmp/snmpd.s1.log -A -c /usr/local/raddle/branch-network/s1.conf	True	51
group-100-linux.class.example.org	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.r2 -Lf /tmp/snmpd.r2.log -A -c /usr/local/raddle/branch-network/r2.conf	True	Up

Figure 13: Status of the snmpd_raddle Process

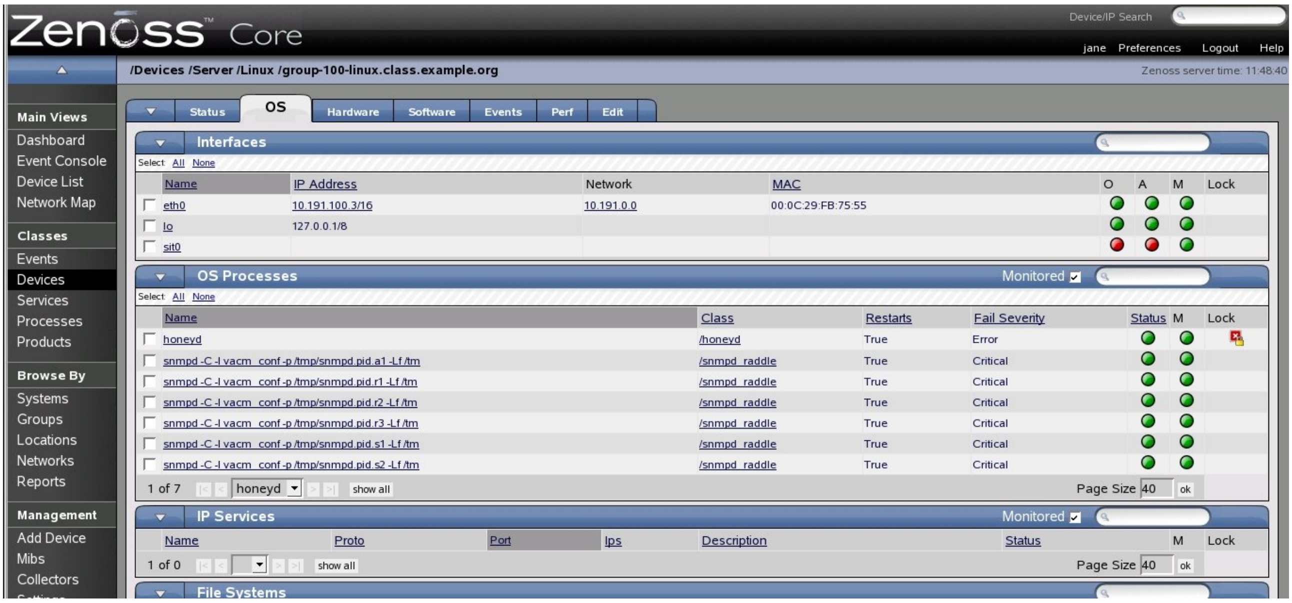
4.2 Process discovery

From a device perspective, the *os* tab allows configuration as to which processes should be monitored and shows their current status. The table drop-down menu allows processes to be added, deleted, locked and monitoring enabled or disabled. This should be used if a process has been configured but with *zMonitor=False*.

Once processes themselves have been configured as described in the previous subsection, then whenever a device is **modelled**, a check will be made for all processes whose *zMonitor* zProperty is set to *True* (either globally or for a specific device). An entry will automatically be added to the Process table under the device's *os* tab for processes that are discovered. By default, zenmodeler runs every 12 hours but any device can be remodelled from the drop-down table menu -> *Manage* -> *Model Device* .

The corollary is also true; if a device remodel takes place and a configured process is **not** running then it is automatically removed from the process section of the *os* tab and monitoring for that process for that device stops, at least until the next remodel.

This can be very inconvenient if an important process happens to be down on the periodic remodel. One way to prevent this hiatus is to select the process for the device and use the table drop-down menu to *Lock from Deletion* . Unfortunately, this sometimes seems to produce adverse effects which result in changes of the process status **not** being monitored.



/Devices /Server /Linux /group-100-linux.class.example.org

Interfaces

	Name	IP Address	Network	MAC	O	A	M	Lock
□	eth0	10.191.100.3/16	10.191.0.0	00:0C:29:FB:75:55	🟢	🟢	🟢	
□	lo	127.0.0.1/8			🟢	🟢	🟢	
□	sit0				🔴	🔴	🟢	

OS Processes

	Name	Class	Restarts	Fail Severity	Status	M	Lock
□	honeyd	/honeyd	True	Error	🟢	🟢	🔒
□	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.a1 -Lf /tm	/snmpd_raddle	True	Critical	🟢	🟢	
□	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.r1 -Lf /tm	/snmpd_raddle	True	Critical	🟢	🟢	
□	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.r2 -Lf /tm	/snmpd_raddle	True	Critical	🟢	🟢	
□	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.r3 -Lf /tm	/snmpd_raddle	True	Critical	🟢	🟢	
□	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.s1 -Lf /tm	/snmpd_raddle	True	Critical	🟢	🟢	
□	snmpd -C -l vacm_conf -p /tmp/snmpd.pid.s2 -Lf /tm	/snmpd_raddle	True	Critical	🟢	🟢	

1 of 7 honeyd Page Size 40

IP Services

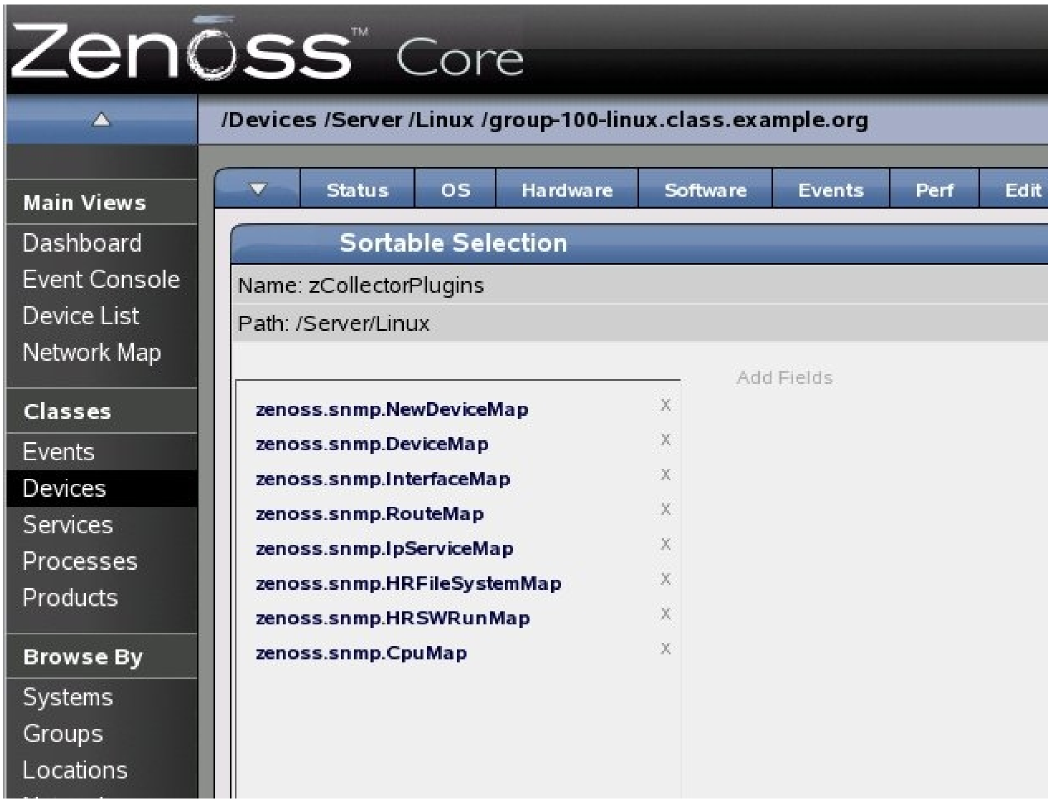
Name	Proto	Port	Ips	Description	Status	M	Lock

1 of 0 Page Size 40

File Systems

Figure 14: Device os tab showing processes with status

Fundamentally, the zenmodeler daemon will use the discovery protocol(s) configured for a device, to discover processes. If the device supports SNMP, then it is usually the Host Resources MIB hrSWRunTable that will provide process information. Modelling collectors for a device are specified from the table drop-down *More -> Collector Plugins* menu. The *zenoss.snmp.HRSWRunMap* is the collector that gather process information from the Host Resources MIB.



Name: zCollectorPlugins

Path: /Server/Linux

zenoss.snmp.NewDeviceMap x
zenoss.snmp.DeviceMap x
zenoss.snmp.InterfaceMap x
zenoss.snmp.RouteMap x
zenoss.snmp.IpServiceMap x
zenoss.snmp.HRFileSystemMap x
zenoss.snmp.HRSWRunMap x
zenoss.snmp.CpuMap x

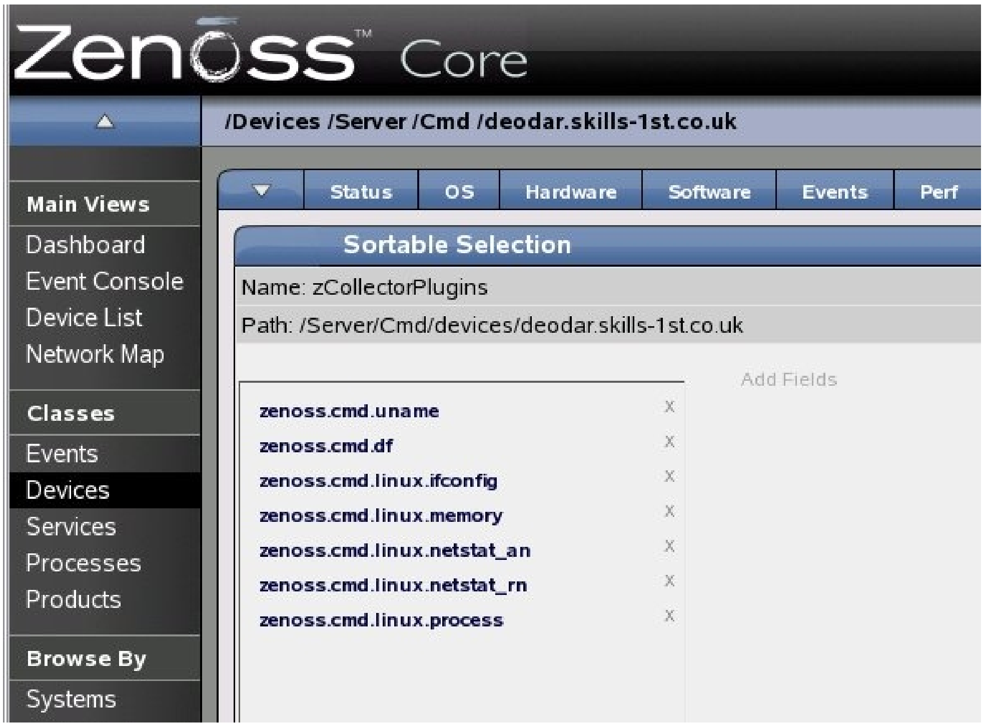
Figure 15: Modelling collector plugins for a device which supports SNMP

To better understand what the modelling process does, try running zenmodeler standalone, with full debugging turned on:

```
zenmodeler run -v 10 -d group-100-linux.class.example.org
```

You should be able to see the process table entries being returned.

For a device that does **not** support SNMP, process modelling can still take place using the *zenoss.cmd.linux.process* modelling collector. Note that these modelling collectors do **not** require the Zenoss plugins to be installed on a remote system – simple operating system commands are run, over SSH, on the remote system (so zProperties need to be configured for a device to permit SSH access)..



Name: zCollectorPlugins

Path: /Server/Cmd/devices/deodar.skills-1st.co.uk

```
zenoss.cmd.uname
zenoss.cmd.df
zenoss.cmd.linux.ifconfig
zenoss.cmd.linux.memory
zenoss.cmd.linux.netstat_an
zenoss.cmd.linux.netstat_rn
zenoss.cmd.linux.process
```

Figure 16: Modelling collector plugins for a non-SNMP device

Again, to better understand what is happening, run zenmodeler with full debugging (*-v 10*) from a command line.

4.3 Process status checking

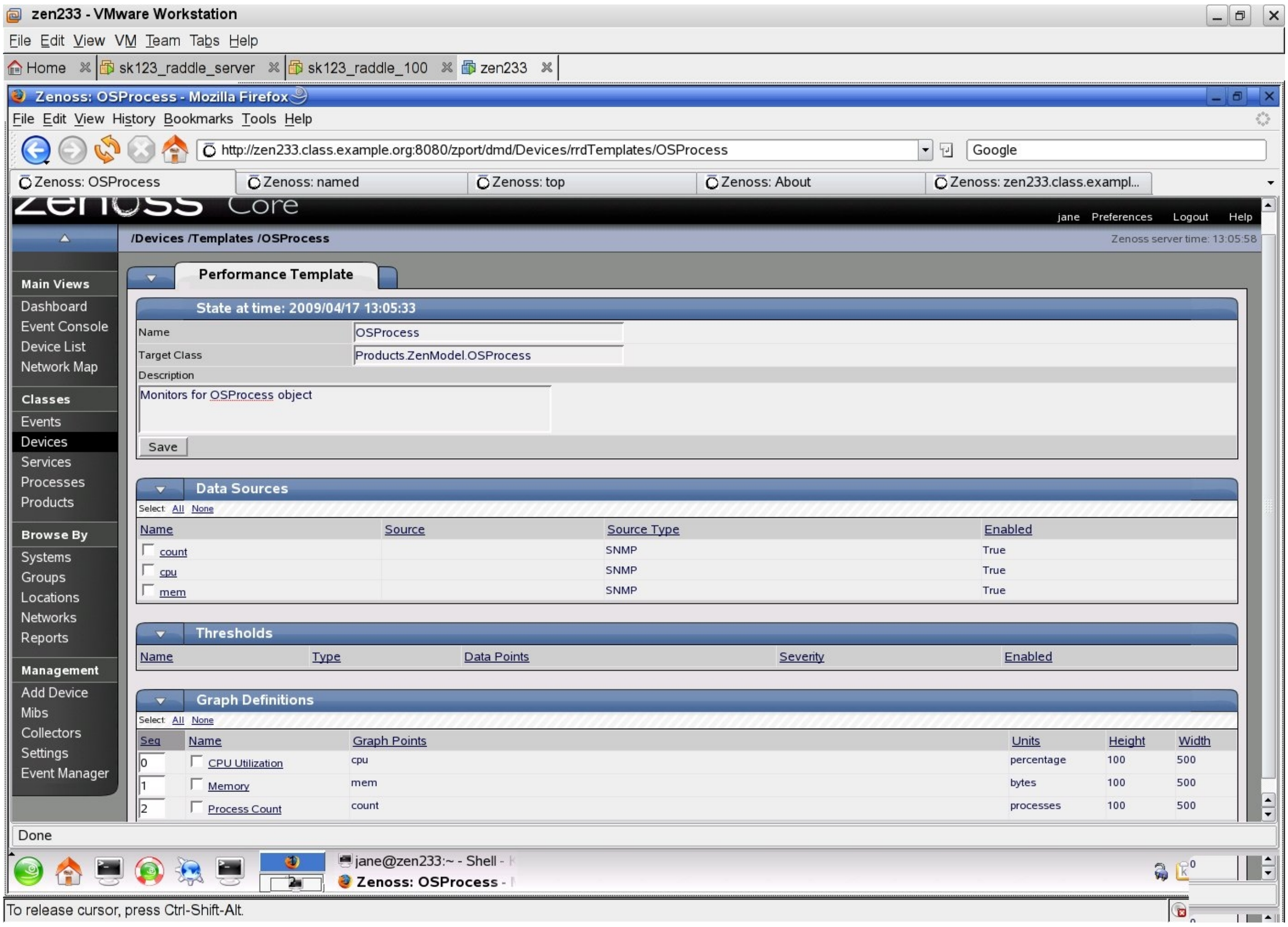
Once processes are discovered for a device (modelled), the zenprocess daemon checks the status of those processes, by default every 3 minutes. The process table in the device's *os* tab should show a green icon for a healthy process and a red icon for a missing process.

Events of the configured severity will be generated when the process is missing and the corresponding cleared event will be generated if *zAlertOnRestart* is set to *True* , when the process is detected.

Note that with Zenoss versions prior to 2.3.3 there was a bug described in TRAC ticket 3270 whereby process status was always reported as up, even when down, but this apparently was only a display problem with the status icon and events were actually still generated accurately.

If the process *Name* field is selected in the *os* tab, then performance data for that process should be displayed. (Note that the *Name* and *Class* columns got swapped around between Zenoss 2.2 and 2.3.).

There is a single performance data collector template, OSProcess, that defines what data to collect. It can be examined by drilling into the performance graphs for a process on a device, and then selecting the *Templates* tab.



State at time: 2009/04/17 13:05:33

Name	OSProcess
Target Class	Products.ZenModel.OSProcess
Description	Monitors for OSProcess object

Data Sources

Select: All None

Name	Source	Source Type	Enabled
□ count		SNMP	True
□ cpu		SNMP	True
□ mem		SNMP	True

Thresholds

Name	Type	Data Points	Severity	Enabled

Graph Definitions

Select: All None

Seq	Name	Graph Points	Units	Height	Width
0	□ CPU Utilization	cpu	percentage	100	500
1	□ Memory	mem	bytes	100	500
2	□ Process Count	count	processes	100	500

Figure 17: OSProcess template for collecting process performance data

The template defines three data sources for:

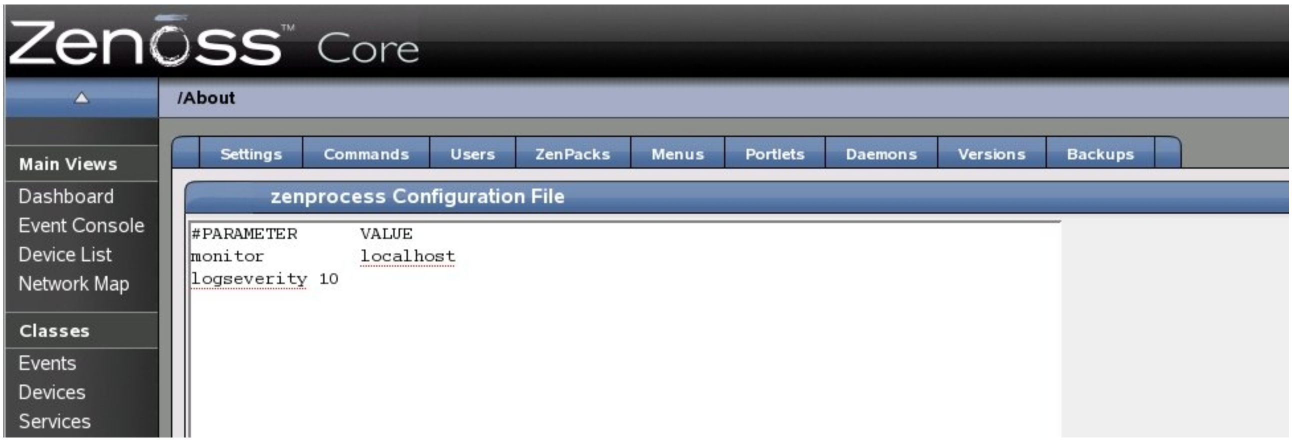
- count (regardless of whether the *zCountProcs* zProperty is True or False)
- CPU
- mem

Each of these data sources apparently are of type SNMP but no OID source is given. Strangely, these graphs **are** populated with data even so; however, if the device has no SNMP access then data is **not** collected (even though the process **modelling** collector can detect the process).

If logging is increased for the zenprocess daemon, it is possible to see that it is actually zenprocess that collects this performance data, not the usual zenperfsnmp daemon. Logging can be increased for any daemon, from the Zenoss GUI, by selecting the left-hand *Settings* menu, choosing the *Daemons* tab and clicking the *edit config* link. Simply add a line with:

```
logseverity 10
```

and restart the daemon from the Daemons tab page.



```
#PARAMETER      VALUE
monitor         localhost
logseverity 10
```

Figure 18: Increasing logging for Zenoss daemons

In summary, Zenoss process monitoring can discover processes on devices and subsequently monitor those processes. With regard to the process management requirements defined at the start of this document, zenprocess monitoring satisfies 1, 3, 4, 5, 6, 7 and 8 to some extent; that is, monitoring for one or more occurrences of a process, based on exact or partial process names and process arguments; by thresholding the process count (which is automatically gathered by zenprocess) then alerts on maximum / minimum numbers of instances of a process can be raised. The zenprocess mechanism not only generates events automatically but can also generate clearing events. Although zenprocess itself cannot take automatic remedial action, the Zenoss event processing subsystem can.

5 Integrating process monitoring with other Zenoss capabilities

So far, a number of different process monitoring techniques have been discussed:

- SNMP using various combinations of MIBs and TRAPs
- SSH to run either Operating System commands or remote scripts
- Nagios plugins
- Zenoss plugins
- Zenoss zenprocess monitoring

The first three techniques don't mandate a Zenoss manager. Strictly the Zenoss plugins could run standalone and deliver output to a different manager; however all these methods integrate well with Zenoss.

5.1 SNMP MIBs, TRAPs and Zenoss

Zenoss has comprehensive facilities to receive and interpret SNMP TRAPs and NOTIFICATIONs (NOTIFICATIONs are effectively SNMP V2 TRAPs and are handled in a similar way by Zenoss; in the ensuing discussion TRAP will be used to embrace both). Some TRAPs are configured when Zenoss is installed (such as warm start, cold start, authentication, link up and link down); any TRAP can be configured through the Zenoss GUI, based on the enterprise OID and the specific TRAP number. All the varbinds on the TRAP are available as user-defined fields on the *Details* tab of a detailed event. By creating **event mappings**, events can be further distinguished using regular expressions to parse the event's *summary* field. Python rules can be used in mappings to test information from the TRAP against other criteria; for example different actions could be taken based on which device sent the TRAP, whether the device is a member of a particular Location or Group and on the Production status of the device.

The TRAP varbinds can also be analysed. Depending on whether criteria are met, an **event mapping transform** can be run – this is typically one or more Python statements that can modify many of the characteristics of both the event and / or the device that generated the event. A simple example would be to change the severity of the event for devices in a particular Group.

For a much more comprehensive discussion, see my Zenoss Event Management paper available at http://www.zenoss.com/Members/jcurry/zenoss_event_management_paper.pdf/view .

The combination in the UCD-SNMP-MIB of process monitoring, the procfix parameter to customise a recovery action, and the ability of the DisMan Event MIB to trigger a recovery action, can interwork with a Zenoss SNMP manager to activate the recovery.

Take the scenario where a process, *named*, has failed and the DisMan Event MIB generates an enterprise specific TRAP to Zenoss, including varbind parameters from the UCD-SNMP-MIB process table. The snmpd.conf configuration file can be seen in Figure 1.

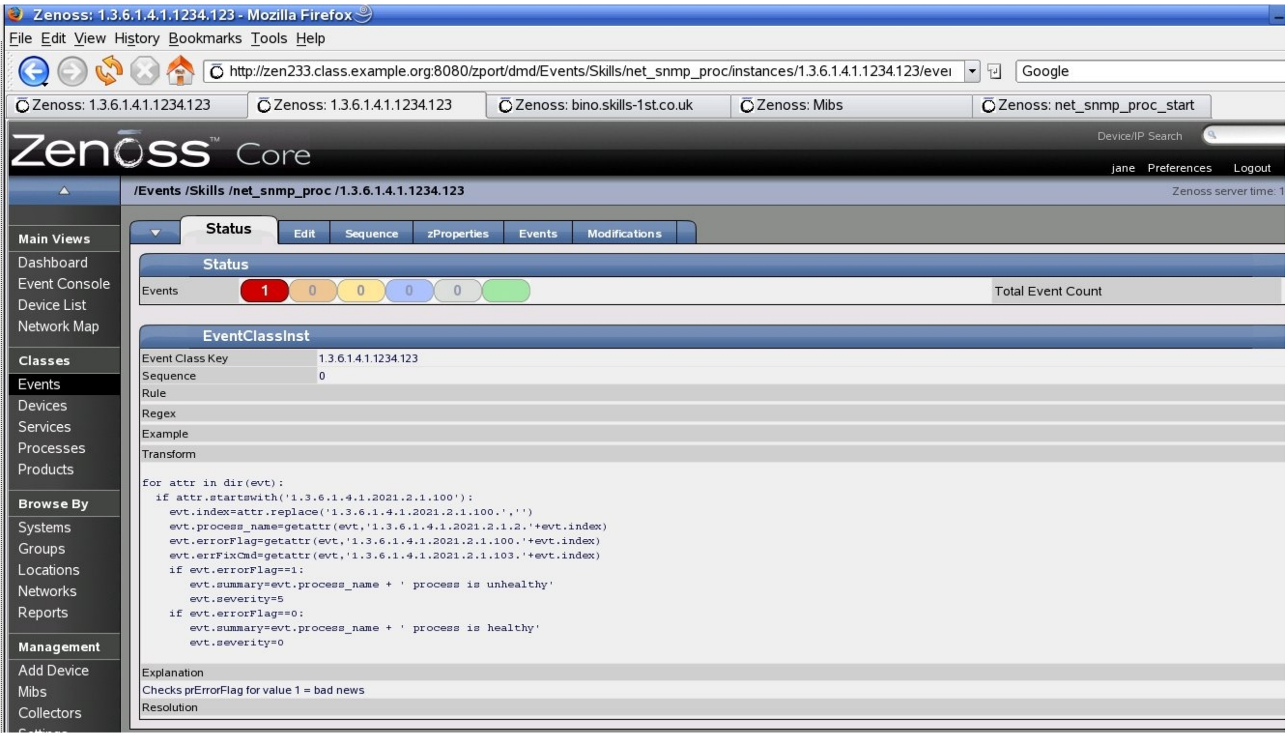
named has a procfix line which specifies to run */etc/init.d/named start* but this **only** happens when the matching instance of *prErrFix* is set to *1*. The monitor line generates an event (strictly an SNMP V2 NOTIFICATION) called *ProcessEvent*,which is defined in the same snmpd.conf (if you don't specify your own event then a default event from the DisMan Event MIB will be sent). The monitor line passes all the parameters for the relevant instance of the UCD-SNMP-MIB process table. The monitor is triggered by the relevant *prErrorFlag != 0*.

- `monitor -u _internal -r 10 -D -S -e ProcessEvent -o prIndex -o prNames -o prMin -o prMax -o prCount -o prErrorFlag -o prErrMessage -o prErrFix -o prErrFixCmd "Process table" prErrorFlag != 0`
- `notificationEvent ProcessEvent .1.3.6.1.4.1.1234.123`

As documented earlier, the net-SNMP agent does not seem able to reliably generate **both** a notification **and** a set event to automatically run a procfix script; hence a Zenoss manager could be used to perform the SNMP SET on the correct *prErrFix* MIB variable. This is probably better practice than having the SNMP agent automatically fix the problem as there will be an audit trail if it is fixed in Zenoss.

5.1.1 Configuring event mapping for SNMP TRAPs

An event mapping should be created for the event generated by the DisMan Event MIB - .1.3.6.1.4.1.1234.123. Start by creating a new event **Class**, whose *eventClassKey* is simply the event OID. In the example below, a new event class, *Skills* is created with an event subclass of *net_snmp_proc*.


Zenoss: 1.3.6.1.4.1.1234.123 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Events/Skills/net_snmp_proc/instances/1.3.6.1.4.1.1234.123/evei
Google
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: bino.skills-1st.co.uk
Zenoss: Mibs
Zenoss: net_snmp_proc_start
Zenoss Core
Device/IP Search
jane
Preferences
Logout
/Events /Skills /net_snmp_proc /1.3.6.1.4.1.1234.123
Zenoss server time: 1
Main Views
Dashboard
Event Console
Device List
Network Map
Classes
Events
Devices
Services
Processes
Products
Browse By
Systems
Groups
Locations
Networks
Reports
Management
Add Device
Mibs
Collectors
Status
Edit
Sequence
zProperties
Events
Modifications
Status
Events
1
0
0
0
0
Total Event Count
EventClassInst

Event Class Key	1.3.6.1.4.1.1234.123
Sequence	0
Rule	
Regex	
Example	
Transform	

```
for attr in dir(evt):
  if attr.startswith('1.3.6.1.4.1.2021.2.1.100'):
    evt.index=attr.replace('1.3.6.1.4.1.2021.2.1.100.','')
    evt.process_name=getattr(evt,'1.3.6.1.4.1.2021.2.1.2.'+evt.index)
    evt.errorFlag=getattr(evt,'1.3.6.1.4.1.2021.2.1.100.'+evt.index)
    evt.errFixCmd=getattr(evt,'1.3.6.1.4.1.2021.2.1.103.'+evt.index)
    if evt.errorFlag==1:
       evt.summary=evt.process_name + ' process is unhealthy'
       evt.severity=5
    if evt.errorFlag==0:
       evt.summary=evt.process_name + ' process is healthy'
       evt.severity=0
```

Explanation

Checks prErrorFlag for value 1 = bad news

Resolution

Figure 19: Event mapping 1.3.6.1.4.1.1234.123 for event class /Skills/net_snmp_proc

Events simply match on the *eventClassKey* of 1.3.6.1.4.1.1234.123 - there is no Rule or Regex matching.

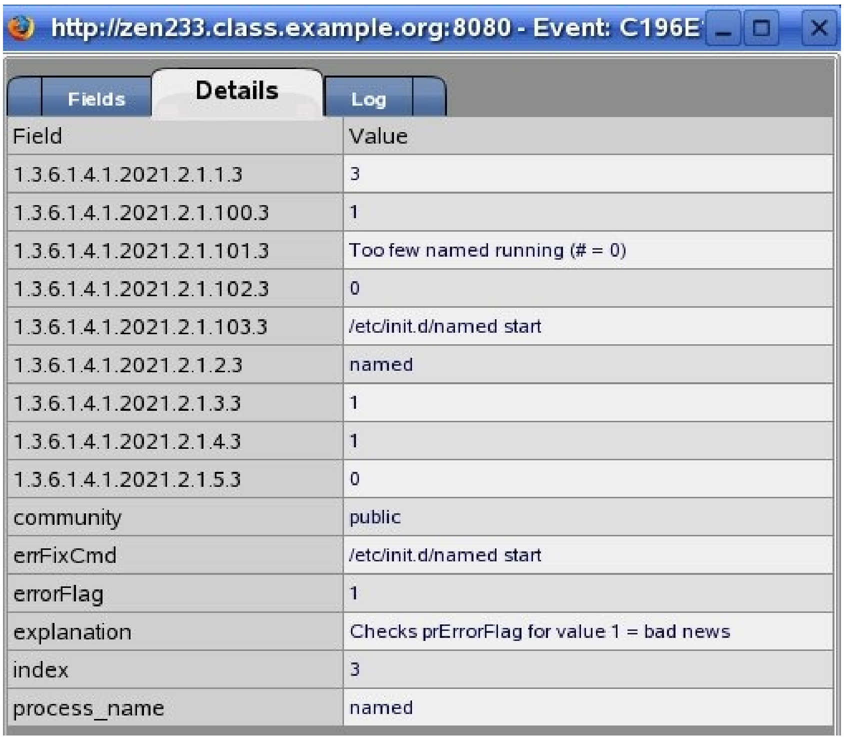
An event mapping transform is applied in order to generate a more useful event summary.

```
for attr in dir(evt):
  if attr.startswith('1.3.6.1.4.1.2021.2.1.100'):
    evt.index=attr.replace('1.3.6.1.4.1.2021.2.1.100.','')
    evt.process_name=getattr(evt,'1.3.6.1.4.1.2021.2.1.2.'+evt.index)
    evt.errorFlag=getattr(evt,'1.3.6.1.4.1.2021.2.1.100.'+evt.index)
    evt.errFixCmd=getattr(evt,'1.3.6.1.4.1.2021.2.1.103.'+evt.index)
    if evt.errorFlag==1:
       evt.summary=evt.process_name + ' process is unhealthy'
       evt.severity=5
```

```
        if evt.errorFlag==0:
           evt.summary=evt.process_name + ' process is healthy'
           evt.severity=0
```

The transform looks for the user-defined event field that represents the *prErrorFlag* varbind (1.3.6.1.4.1.2021.2.1.100). Remember that the UCD-SNMP-MIB has a **table** associated with processes – we need to get at the **index** into that table, which is the last number of the OID, so the transform gets the index into user-defined event field, *evt.Index,* the process name into *evt.Process_name* and the error flag into *evt.errorFlag*. The transform also gets the *prErrFixCmd* value although it is not actually used.

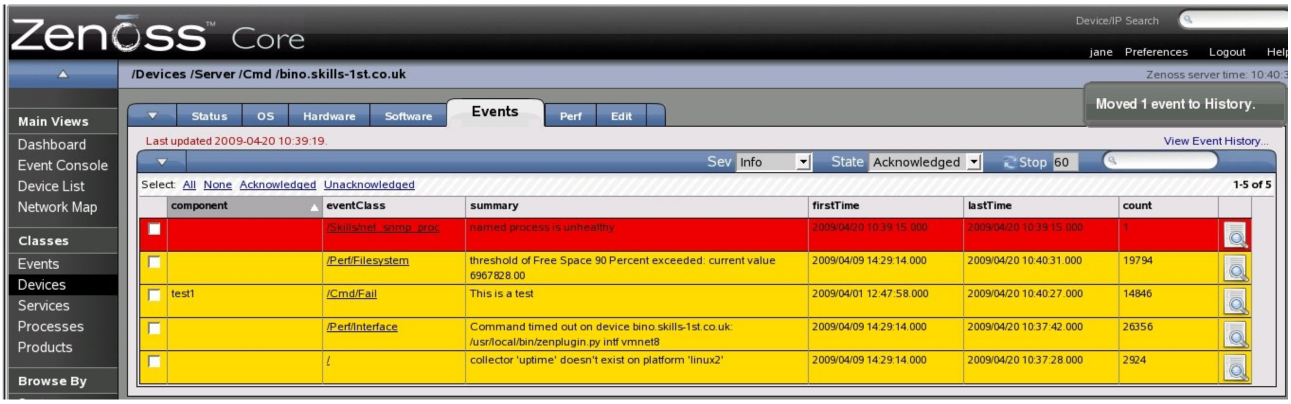
A test then checks *evt.errorFlag*. For a "bad news" event, the summary is set to a useful comment and the severity is set to Critical; for a "good news" event, the severity is set to Cleared. This means that Zenoss's automatic "good news clears bad news" logic will apply.



Field	Value
1.3.6.1.4.1.2021.2.1.1.3	3
1.3.6.1.4.1.2021.2.1.100.3	1
1.3.6.1.4.1.2021.2.1.101.3	Too few named running (# = 0)
1.3.6.1.4.1.2021.2.1.102.3	0
1.3.6.1.4.1.2021.2.1.103.3	/etc/init.d/named start
1.3.6.1.4.1.2021.2.1.2.3	named
1.3.6.1.4.1.2021.2.1.3.3	1
1.3.6.1.4.1.2021.2.1.4.3	1
1.3.6.1.4.1.2021.2.1.5.3	0
community	public
errFixCmd	/etc/init.d/named start
errorFlag	1
explanation	Checks prErrorFlag for value 1 = bad news
index	3
process_name	named

Figure 20: Details tab of event detail for SNMP TRAP 1.3.6.1.4.1.1234.123 showing TRAP varbinds

The resulting Zenoss event appears as shown in the next Figure.



/Devices /Server /Cmd /bino.skills-1st.co.uk

Moved 1 event to History.

Last updated 2009-04-20 10:39:19.

Sev Info State Acknowledged Stop 60

1-5 of 5

	component	eventClass	summary	firstTime	lastTime	count
□		/Skills/net_snmp_proc	named process is unhealthy	2009/04/20 10:39:15.000	2009/04/20 10:39:15.000	1
□		/Perf/Filesystem	threshold of Free Space 90 Percent exceeded: current value 6967828.00	2009/04/09 14:29:14.000	2009/04/20 10:40:31.000	19794
□	test1	/Cmd/Fail	This is a test	2009/04/01 12:47:58.000	2009/04/20 10:40:27.000	14846
□		/Perf/Interface	Command timed out on device bino.skills-1st.co.uk: /usr/local/bin/zenplugin.py intf vmnet8	2009/04/09 14:29:14.000	2009/04/20 10:37:42.000	26356
□		/	collector 'uptime' doesn't exist on platform 'linux2'	2009/04/09 14:29:14.000	2009/04/20 10:37:28.000	2924

Figure 21: "Bad news" event from net-SNMP agent for named process

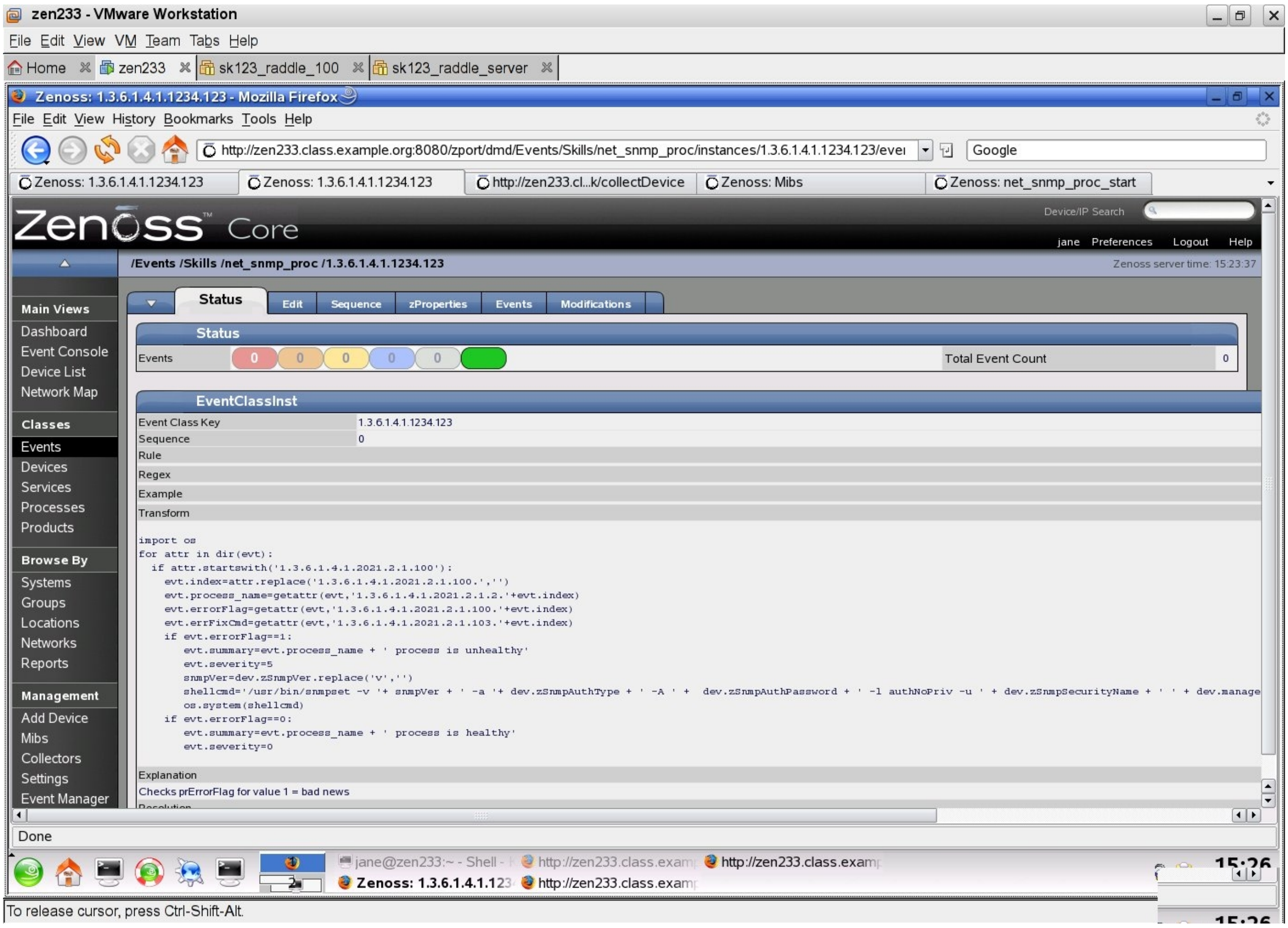
As can be seen from Figure 20, the SNMP TRAP varbinds include the procfix *prErrFixCmd* parameter */etc/init.d/named start* as OID .1.3.6.1.4.1.2021.2.1.103.3 and the status of the trigger, OID .1.3.6.1.4.1.2021.2.1.102.3, the *prErrFix* flag.

5.1.2 Responding to SNMP TRAPs with Zenoss

To automate recovery from process failure using Zenoss, the relevant *prErrFix* flag needs to be set to 1 using SNMP. Bear in mind that this will use an SNMP SET command so SNMP authentication must permit SETs as well as GETs.

One way to configure Zenoss responses is to create Event Commands which are run by the *zenactions* daemon; however, our response needs access to the TRAP varbinds to determine the prTable table index and to set the appropriate *prErrFix* OID variable, and unfortunately, Zenoss Event Commands do not have access to user-defined event fields (ie. the varbinds).

For this reason, the SNMP SET command will be run by extending the event mapping transform given in Figure 19. Any Python program can call Operating System commands (and that's all an event transform is!). To use such commands the *os* Python module needs to be imported, the command text needs to be setup and then the *os.system* method is called.


zen233 - VMware Workstation
File Edit View VM Team Tabs Help
Home
zen233
sk123_raddle_100
sk123_raddle_server
Zenoss: 1.3.6.1.4.1.1234.123 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Events/Skills/net_snmp_proc/instances/1.3.6.1.4.1.1234.123/evei
Google
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: 1.3.6.1.4.1.1234.123
http://zen233.cl...k/collectDevice
Zenoss: Mibs
Zenoss: net_snmp_proc_start
Zenoss Core
Device/IP Search
jane Preferences Logout Help
/Events /Skills /net_snmp_proc /1.3.6.1.4.1.1234.123
Zenoss server time: 15:23:37
Main Views
Dashboard
Event Console
Device List
Network Map
Classes
Events
Devices
Services
Processes
Products
Browse By
Systems
Groups
Locations
Networks
Reports
Management
Add Device
Mibs
Collectors
Settings
Event Manager
Status
Edit
Sequence
zProperties
Events
Modifications
Done
jane@zen233:~ - Shell
http://zen233.class.examp
http://zen233.class.examp
Zenoss: 1.3.6.1.4.1.123
http://zen233.class.examp
15:26
To release cursor, press Ctrl-Shift-Alt.

Status

Events	0	0	0	0	0	🟩	Total Event Count	0

EventClassInst

Event Class Key	1.3.6.1.4.1.1234.123
Sequence	0
Rule	
Regex	
Example	
Transform	

```
import os
for attr in dir(evt):
  if attr.startswith('1.3.6.1.4.1.2021.2.1.100'):
    evt.index=attr.replace('1.3.6.1.4.1.2021.2.1.100.','')
    evt.process_name=getattr(evt,'1.3.6.1.4.1.2021.2.1.2.'+evt.index)
    evt.errorFlag=getattr(evt,'1.3.6.1.4.1.2021.2.1.100.'+evt.index)
    evt.errFixCmd=getattr(evt,'1.3.6.1.4.1.2021.2.1.103.'+evt.index)
    if evt.errorFlag==1:
       evt.summary=evt.process_name + ' process is unhealthy'
       evt.severity=5
       snmpVer=dev.zSnmpVer.replace('v','')
       shellcmd='/usr/bin/snmpset -v '+ snmpVer + ' -a '+ dev.zSnmpAuthType + ' -A ' +  dev.zSnmpAuthPassword + ' -l authNoPriv -u ' + dev.zSnmpSecurityName + ' ' + dev.manage
       os.system(shellcmd)
    if evt.errorFlag==0:
       evt.summary=evt.process_name + ' process is healthy'
       evt.severity=0
```

Explanation

Checks prErrorFlag for value 1 = bad news

Figure 22: Event mapping transform including action to SET the correct prErrFix variable to trigger process restart

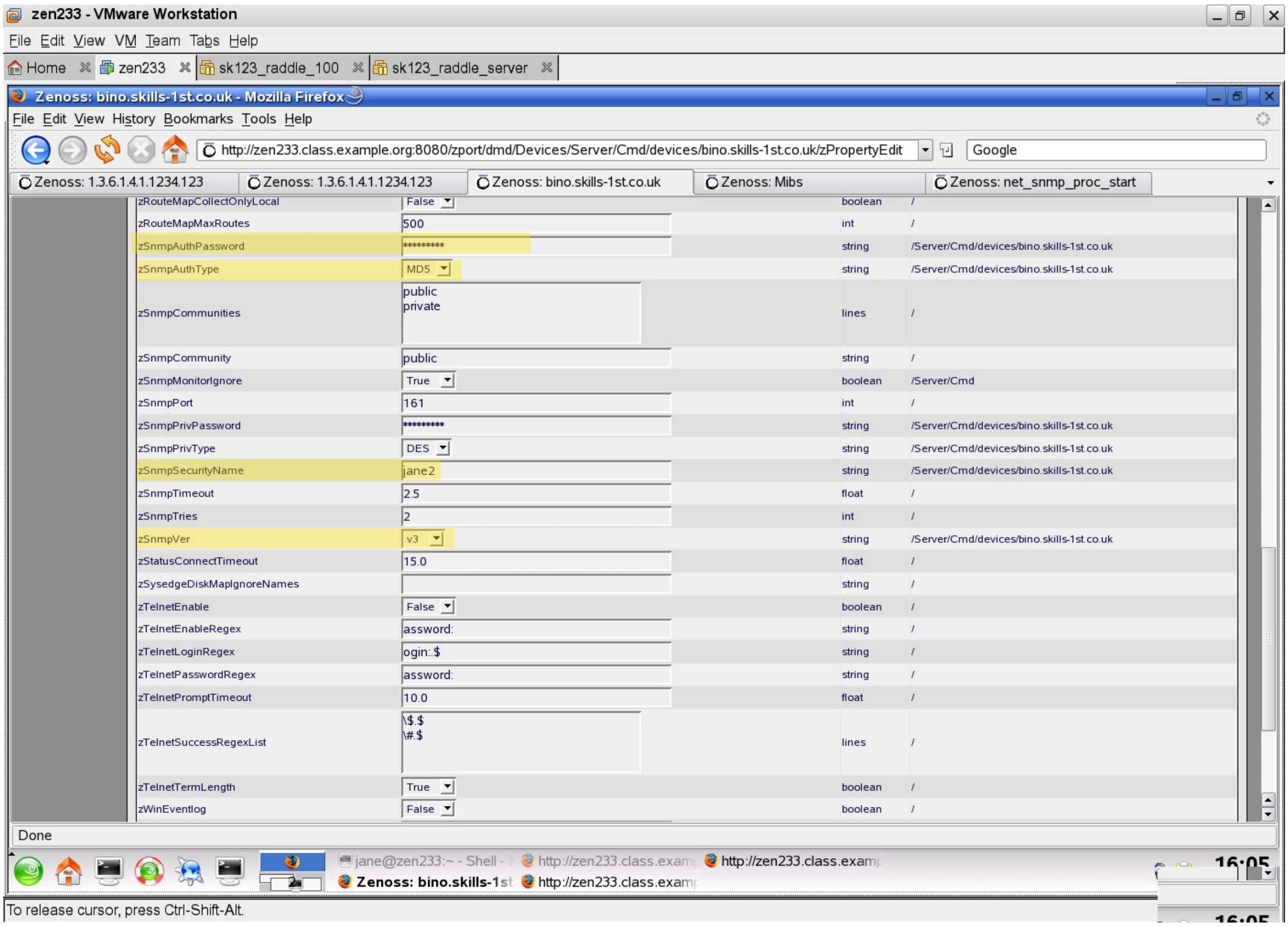
Note that the shell command should all be on one line.

```
import os

                ......

snmpVer=dev.zSnmpVer.replace('v','')

shellcmd='/usr/bin/snmpset -v '+ snmpVer + ' -a '+ dev.zSnmpAuthType + ' -A ' +  dev.zSnmpAuthPassword + ' -l authNoPriv -u ' + dev.zSnmpSecurityName + ' ' + dev.manageIp + ' 1.3.6.1.4.1.2021.2.1.102.'+evt.index+' i 1

os.system(shellcmd)
```

The shell command simply invokes the snmpset command. The example above is for a class of devices that support SNMP V3 so the authentication type, the authentication password and the SNMP V3 user name must be supplied as parameters to snmpset. Rather than hard-code these, they can be accessed from the zProperties of the device that raised the initial TRAP, along with the IP address of that device, and the version of SNMP to use. The only"gotcha" is that the zSnmpVer zProperty responds with **v**3 (in this case) – the snmpset command requires a *-v* parameter followed by a space and a version (1, 2c, 3) so an extra step is shown which strips the leading **v** off the zSnmpVer zProperty.

The end of the snmpset command concatenates the OID for the *prErrFix* variable with the correct index from the user-defined *evt.index* value and sets the value, of type *I* (INTEGER) to the value *1* – in other words, run the configured *prErrFixCmd*, */etc/init.d/named start* .

Do ensure that Zenoss has been configured correctly with SNMP zProperties for devices and / or device classes.


zen233 - VMware Workstation
File Edit View VM Team Tabs Help
Home
zen233
sk123_raddle_100
sk123_raddle_server
Zenoss: bino.skills-1st.co.uk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Devices/Server/Cmd/devices/bino.skills-1st.co.uk/zPropertyEdit
Google
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: bino.skills-1st.co.uk
Zenoss: Mibs
Zenoss: net_snmp_proc_start
Done
jane@zen233:~ - Shell -
http://zen233.class.examp
http://zen233.class.examp
Zenoss: bino.skills-1st
http://zen233.class.examp
To release cursor, press Ctrl-Shift-Alt.
16:05

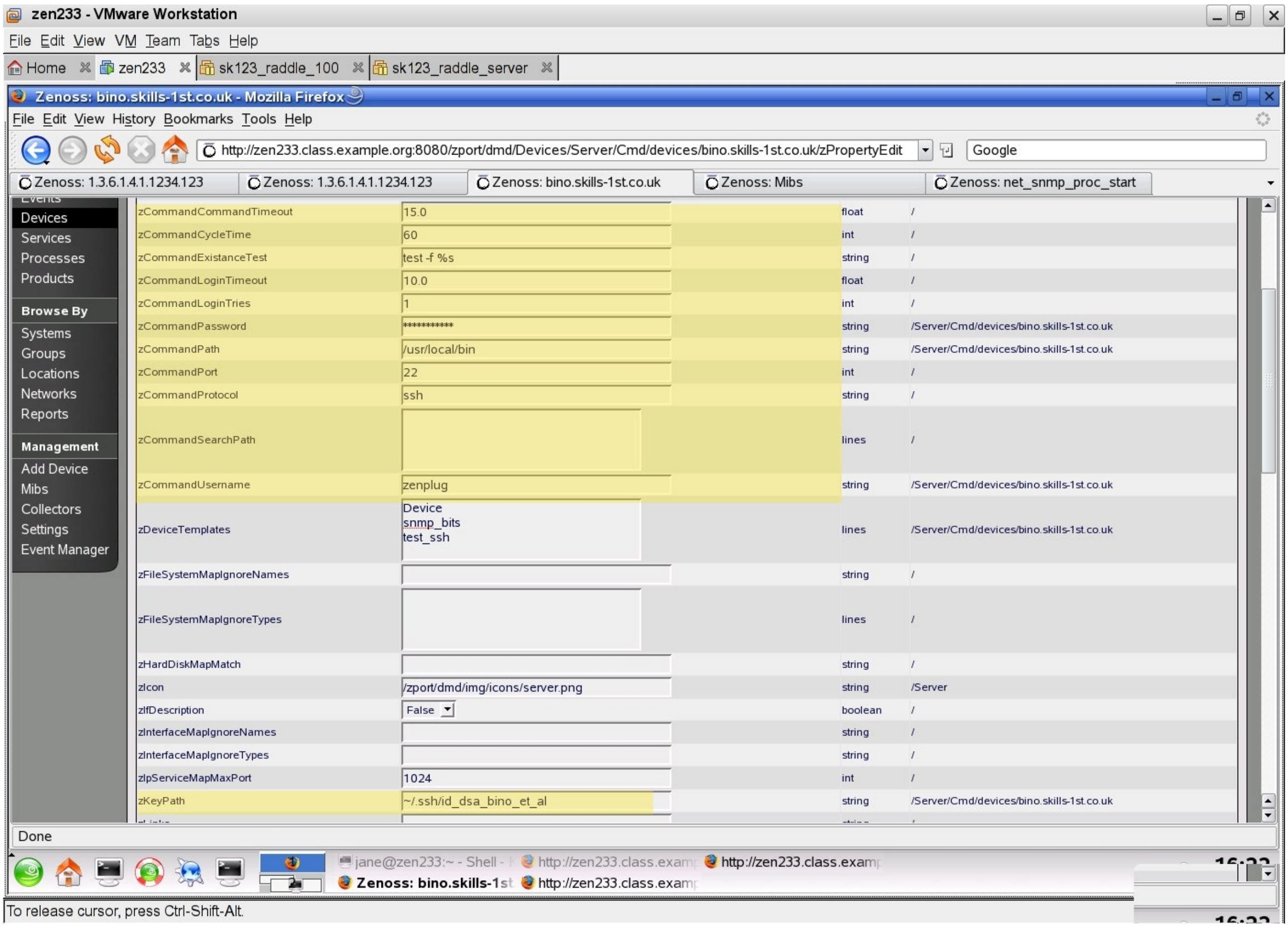
zRouteMapCollectOnlyLocal	False	boolean	/
zRouteMapMaxRoutes	500	int	/
zSnmpAuthPassword	*********	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zSnmpAuthType	MD5	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zSnmpCommunities	public private	lines	/
zSnmpCommunity	public	string	/
zSnmpMonitorIgnore	True	boolean	/Server/Cmd
zSnmpPort	161	int	/
zSnmpPrivPassword	*********	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zSnmpPrivType	DES	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zSnmpSecurityName	jane2	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zSnmpTimeout	2.5	float	/
zSnmpTries	2	int	/
zSnmpVer	v3	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zStatusConnectTimeout	15.0	float	/
zSysedgeDiskMapIgnoreNames		string	/
zTelnetEnable	False	boolean	/
zTelnetEnableRegex	assword:	string	/
zTelnetLoginRegex	ogin:.$	string	/
zTelnetPasswordRegex	assword:	string	/
zTelnetPromptTimeout	10.0	float	/
zTelnetSuccessRegexList	\$.$ \#.$	lines	/
zTelnetTermLength	True	boolean	/
zWinEventlog	False	boolean	/

Figure 23: Zenoss SNMP zProperties for an SNMP V3 device class

In practise, all this explanation takes far longer than the automation does!

5.2 Zenoss and SSH

Each device class and / or specific device can have zProperties configured for SSH communications. Once accomplished, any underlying Zenoss SSH commands will simply use those parameters.


zen233 - VMware Workstation
File Edit View VM Team Tabs Help
Home
zen233
sk123_raddle_100
sk123_raddle_server
Zenoss: bino.skills-1st.co.uk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Devices/Server/Cmd/devices/bino.skills-1st.co.uk/zPropertyEdit
Google
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: 1.3.6.1.4.1.1234.123
Zenoss: bino.skills-1st.co.uk
Zenoss: Mibs
Zenoss: net_snmp_proc_start
Events
Devices
Services
Processes
Products
Browse By
Systems
Groups
Locations
Networks
Reports
Management
Add Device
Mibs
Collectors
Settings
Event Manager
Done
jane@zen233:~ - Shell -
http://zen233.class.exam
http://zen233.class.exam
Zenoss: bino.skills-1st
http://zen233.class.exam
To release cursor, press Ctrl-Shift-Alt.

zCommandCommandTimeout	15.0	float	/
zCommandCycleTime	60	int	/
zCommandExistanceTest	test -f %s	string	/
zCommandLoginTimeout	10.0	float	/
zCommandLoginTries	1	int	/
zCommandPassword	***********	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zCommandPath	/usr/local/bin	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zCommandPort	22	int	/
zCommandProtocol	ssh	string	/
zCommandSearchPath		lines	/
zCommandUsername	zenplug	string	/Server/Cmd/devices/bino.skills-1st.co.uk
zDeviceTemplates	Device snmp_bits test_ssh	lines	/Server/Cmd/devices/bino.skills-1st.co.uk
zFileSystemMapIgnoreNames		string	/
zFileSystemMapIgnoreTypes		lines	/
zHardDiskMapMatch		string	/
zIcon	/zport/dmd/img/icons/server.png	string	/Server
zIfDescription	False	boolean	/
zInterfaceMapIgnoreNames		string	/
zInterfaceMapIgnoreTypes		string	/
zIpServiceMapMaxPort	1024	int	/
zKeyPath	~/.ssh/id_dsa_bino_et_al	string	/Server/Cmd/devices/bino.skills-1st.co.uk

Figure 24: Zenoss SSH zProperties for device class

The crucial parameters are:

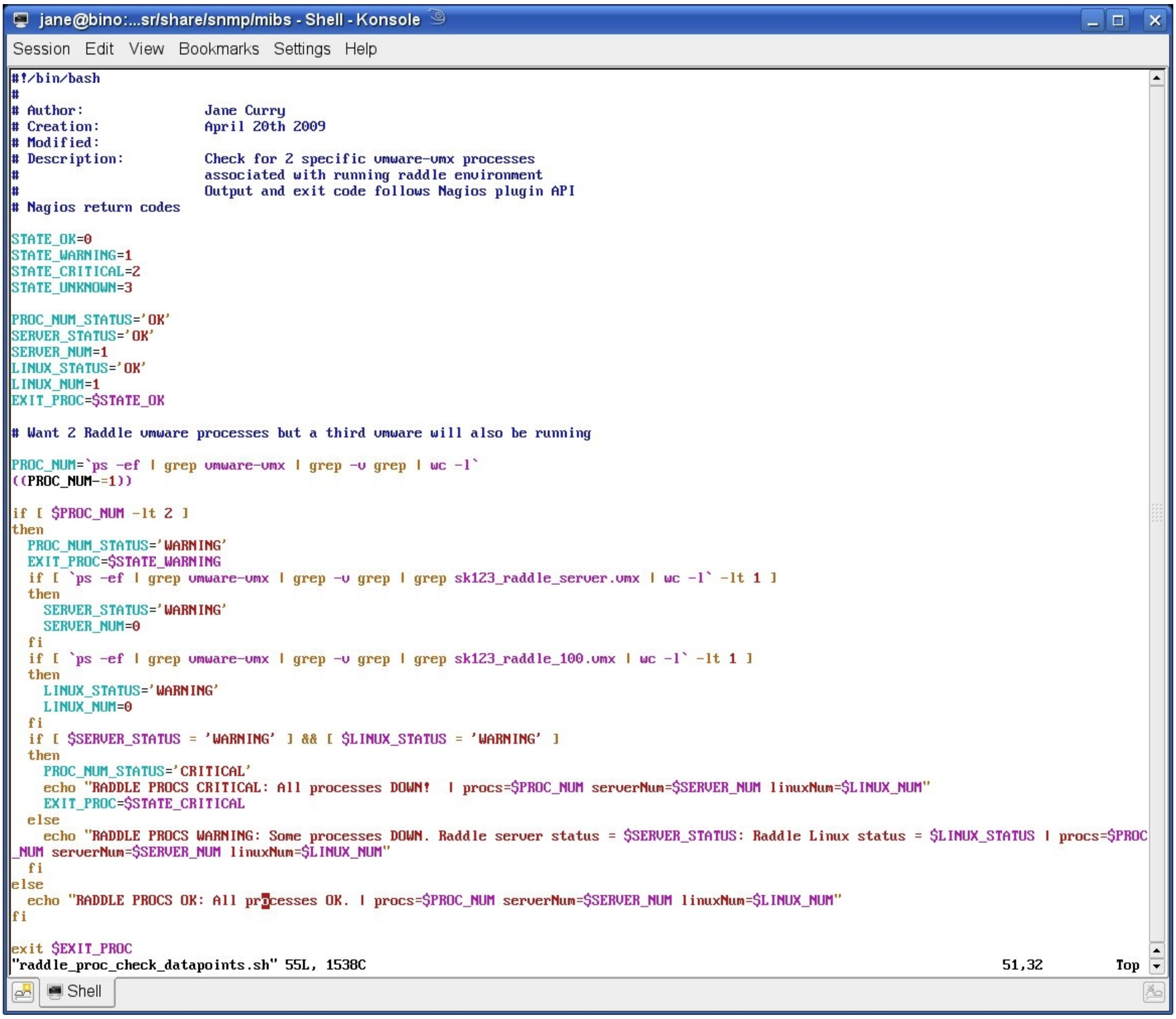
- zCommandPassword this is the passphrase if one was defined
- zCommandPath path for remote commands
- zCommandSearchPath path for remote commands (**Note** that this currently seems to have no effect)
- zCommandUsername the username already setup for SSH
- zKeyPath where the SSH private key file is

Note that the screenshot above demonstrates the possibility of using a non-standard name for the key file, *id_dsa_bino_et_al*. This file should be in the zenoss user's .ssh directory.

Note that if non-standard keyfile names are used, Zenoss appears to need the public key file (*id_dsa_bino_et_al.pub*) in the .ssh directory, in addition to the private key file.

5.2.1 Using Zenoss to run stand-alone SSH commands

Any command can potentially be run on a remote system using SSH. If a specific combination of processes is required to define a "healthy" service, then a script may be the easiest way to accomplish this. As a simple example, consider the script below:



```
#!/bin/bash
#
# Author:                Jane Curry
# Creation:              April 20th 2009
# Modified:
# Description:           Check for 2 specific vmware-vmx processes
#                        associated with running raddle environment
#                        Output and exit code follows Nagios plugin API
# Nagios return codes

STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3

PROC_NUM_STATUS='OK'
SERVER_STATUS='OK'
SERVER_NUM=1
LINUX_STATUS='OK'
LINUX_NUM=1
EXIT_PROC=$STATE_OK

# Want 2 Raddle vmware processes but a third vmware will also be running

PROC_NUM=`ps -ef | grep vmware-vmx | grep -v grep | wc -l`
((PROC_NUM-=1))

if [ $PROC_NUM -lt 2 ]
then
  PROC_NUM_STATUS='WARNING'
  EXIT_PROC=$STATE_WARNING
  if [ `ps -ef | grep vmware-vmx | grep -v grep | grep sk123_raddle_server.vmx | wc -l` -lt 1 ]
  then
    SERVER_STATUS='WARNING'
    SERVER_NUM=0
  fi
  if [ `ps -ef | grep vmware-vmx | grep -v grep | grep sk123_raddle_100.vmx | wc -l` -lt 1 ]
  then
    LINUX_STATUS='WARNING'
    LINUX_NUM=0
  fi
  if [ $SERVER_STATUS = 'WARNING' ] && [ $LINUX_STATUS = 'WARNING' ]
  then
    PROC_NUM_STATUS='CRITICAL'
    echo "RADDLE PROCS CRITICAL: All processes DOWN!  | procs=$PROC_NUM serverNum=$SERVER_NUM linuxNum=$LINUX_NUM"
    EXIT_PROC=$STATE_CRITICAL
  else
    echo "RADDLE PROCS WARNING: Some processes DOWN. Raddle server status = $SERVER_STATUS: Raddle Linux status = $LINUX_STATUS | procs=$PROC
_NUM serverNum=$SERVER_NUM linuxNum=$LINUX_NUM"
  fi
else
  echo "RADDLE PROCS OK: All processes OK. | procs=$PROC_NUM serverNum=$SERVER_NUM linuxNum=$LINUX_NUM"
fi

exit $EXIT_PROC
"raddle_proc_check_datapoints.sh" 55L, 1538C                                              51,32         Top
```

Figure 25: Shellscript to check for specific processes

The script is checking for two VMware processes, one for a machine called server, the other for a machine called group-100-linux; these two VMs together make up the **raddle** application. The script will return numeric values for the number of relevant VMware processes, the number of "server" processes and the number of "linux" processes. The exit code will be OK if both are running, WARNING if only 1 is running and CRITICAL if both are down. No attempt is made in this script to rectify any problem, but potentially, recovery actions could also be included.

This script uses elements of the Nagios API to return a single line of output with:

- The status of the script, followed by colon, followed by textual information

- A vertical bar
- Performance data in the format *label=value* . Multiple entries are space-separated

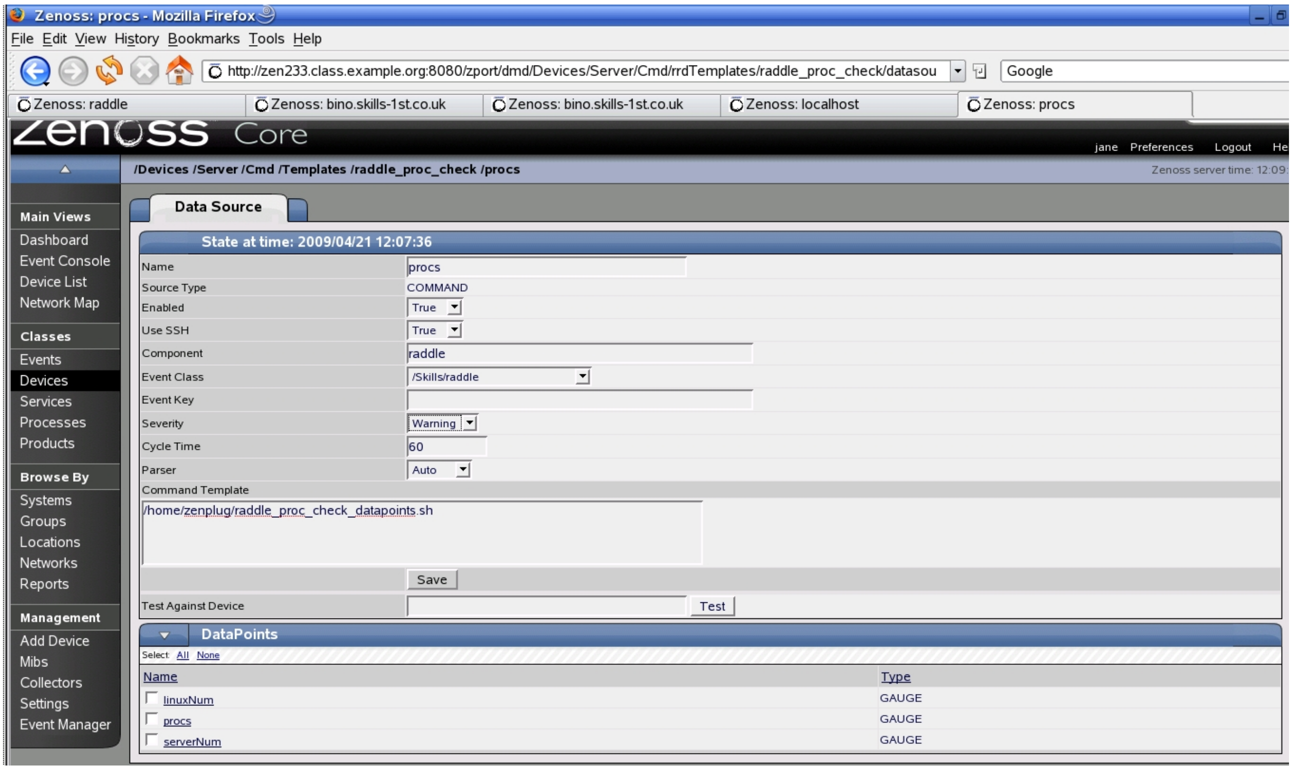
The script also returns an exit status as defined by Nagios – 0 = OK, 1 = WARNING, 2 = CRITICAL, 3 = UNKNOWN.

To make use of a command script, the easiest method is to setup a Zenoss performance data collector **template**. Note that it is good practice to create templates at a device **class** level – otherwise, if it is created for a specific device, there is no simple way to later apply that template to other devices. Data is actually collected by Zenoss's **zencommand** daemon.

A performance data collection template has a number of elements:

● Data Sources	**how** to collect data
● Thresholds	ranges for “healthy” data
● Graph Definitions	**what** to plot and **how** to plot it

The Data Source specifies what command to run, where to run it, and how to run it.



/Devices /Server /Cmd /Templates /raddle_proc_check /procs

State at time: 2009/04/21 12:07:36

Name	procs
Source Type	COMMAND
Enabled	True
Use SSH	True
Component	raddle
Event Class	/Skills/raddle
Event Key	
Severity	Warning
Cycle Time	60
Parser	Auto
Command Template	/home/zenplug/raddle_proc_check_datapoints.sh
Test Against Device	

DataPoints

Name	Type
☐ linuxNum	GAUGE
☐ procs	GAUGE
☐ serverNum	GAUGE

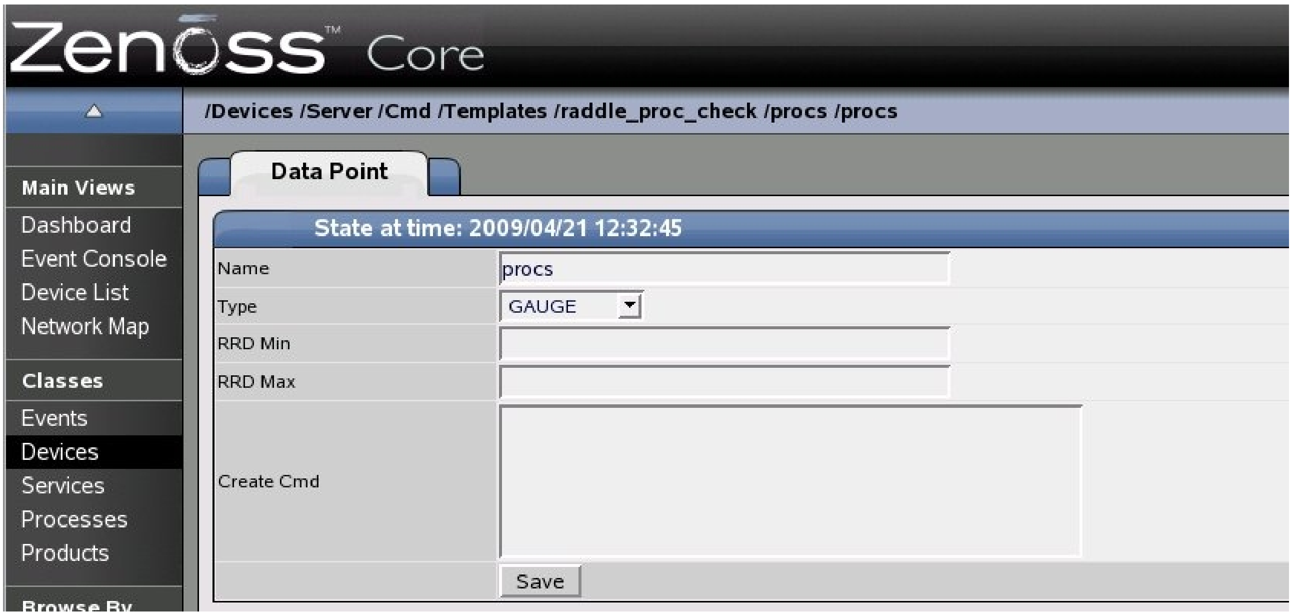
Figure 26: Defining the procs Data Source in the raddle_proc_check performance data collector template

In the Data Source dialogue:

- Source Type should be *COMMAND*. The drop down will certainly offer SNMP as another alternative. If other ZenPacks are installed then other types may also be available.
- To use this data source on remote systems over SSH, ensure the *Use SSH* box is *True*
- The *Component* field is useful when processing events – for example, it is one of the fields used to determine whether an event is a duplicate. The component field does not need to already exist anywhere else – it is simply a text string. *raddle* has been used here.
- The *Event Class* field will default to */Cmd/Fail* but could usefully be set to an existing, locally-defined event class. Here the class is set to */Skills/raddle*.
- The *Cycle Time* is how frequently the zencommand daemon will run the script.
- The *Command Template* is the script you want to run. If a fully-qualified pathname is provided then it will be honoured; otherwise, zencommand will consult the zProperties for a device and will prepend the zCommandPath to the filename given in the *Command Template*.
- Don't forget to use the *Save* button after completing definitions
- **Note** that the *Test* button does not appear to work for invoking remote commands. It returns a "No such file or directory" error. Similarly the *zentestcommand* utility returns the same error for remote scripts.
- The easiest way to test the script over SSH is to run the zencommand with full debug; for example:

```
      zencommand run -v 10 -d bino.skills-1st.co.uk
```

The bottom part of the Data Source dialogue maps the data that the script collects into Zenoss DataPoints that can be thresholded and graphed. Remember that the script in Figure 25 delivered three data values after the vertical bar on the output line – procs, serverNum and linuxNum. **The definitions of the DataPoints must match these label names exactly.**



/Devices /Server /Cmd /Templates /raddle_proc_check /procs /procs

State at time: 2009/04/21 12:32:45

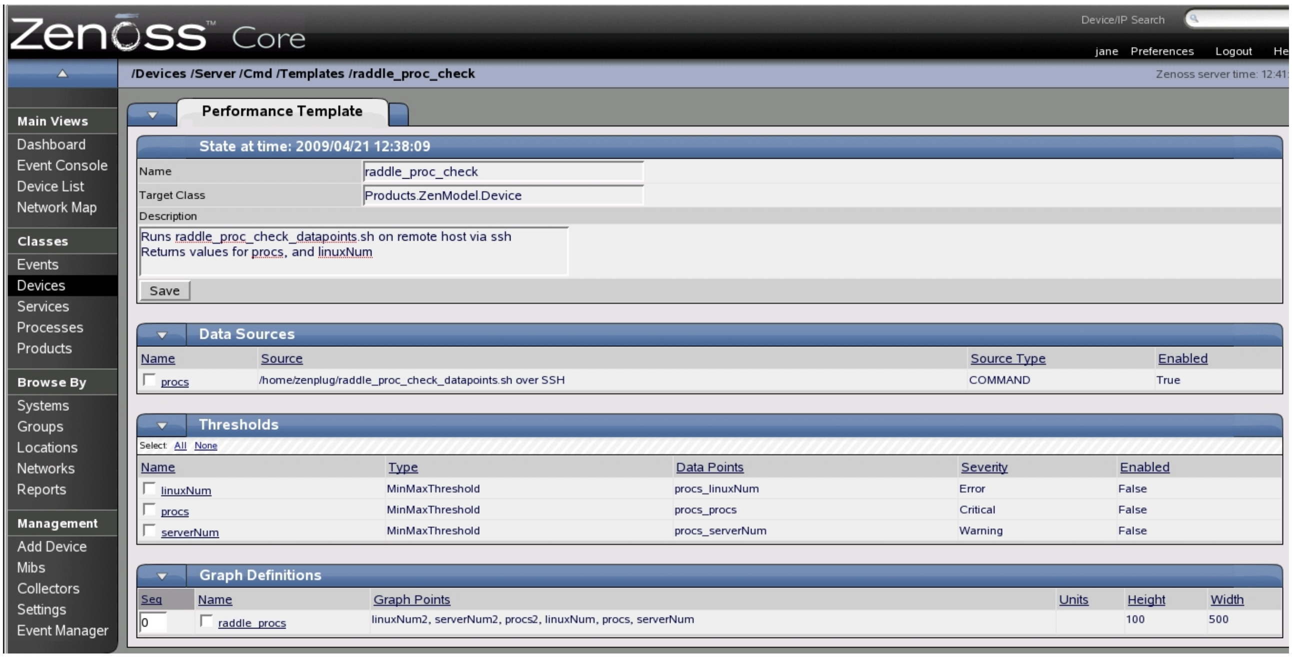
Name	procs
Type	GAUGE
RRD Min	
RRD Max	
Create Cmd	

Figure 27: Defining the procs DataPoint in the procs Data Source

Typically, DataPoint definitions can be left at defaults having ensured that the name matches the label that the script delivers.

The Zenoss name for a DataPoint is the concatenation of the Data Source and the DataPoint names; hence, in the screenshot above, the DataPoint is *procs_procs*. The other two DataPoints will be *procs_serverNum* and *procs_linuxNum*. For this reason, it is important **not** to change the name of the Data Source without due consideration or DataPoints already used in graphs and thresholds will become undefined.

Once the Data Source and DataPoints are defined, thresholds and graphs can be setup within the template.



/Devices /Server /Cmd /Templates /raddle_proc_check

Zenoss server time: 12:41

State at time: 2009/04/21 12:38:09

Name	raddle_proc_check
Target Class	Products.ZenModel.Device
Description	Runs raddle_proc_check_datapoints.sh on remote host via ssh Returns values for procs, and linuxNum

Data Sources

Name	Source	Source Type	Enabled
□ procs	/home/zenplug/raddle_proc_check_datapoints.sh over SSH	COMMAND	True

Thresholds

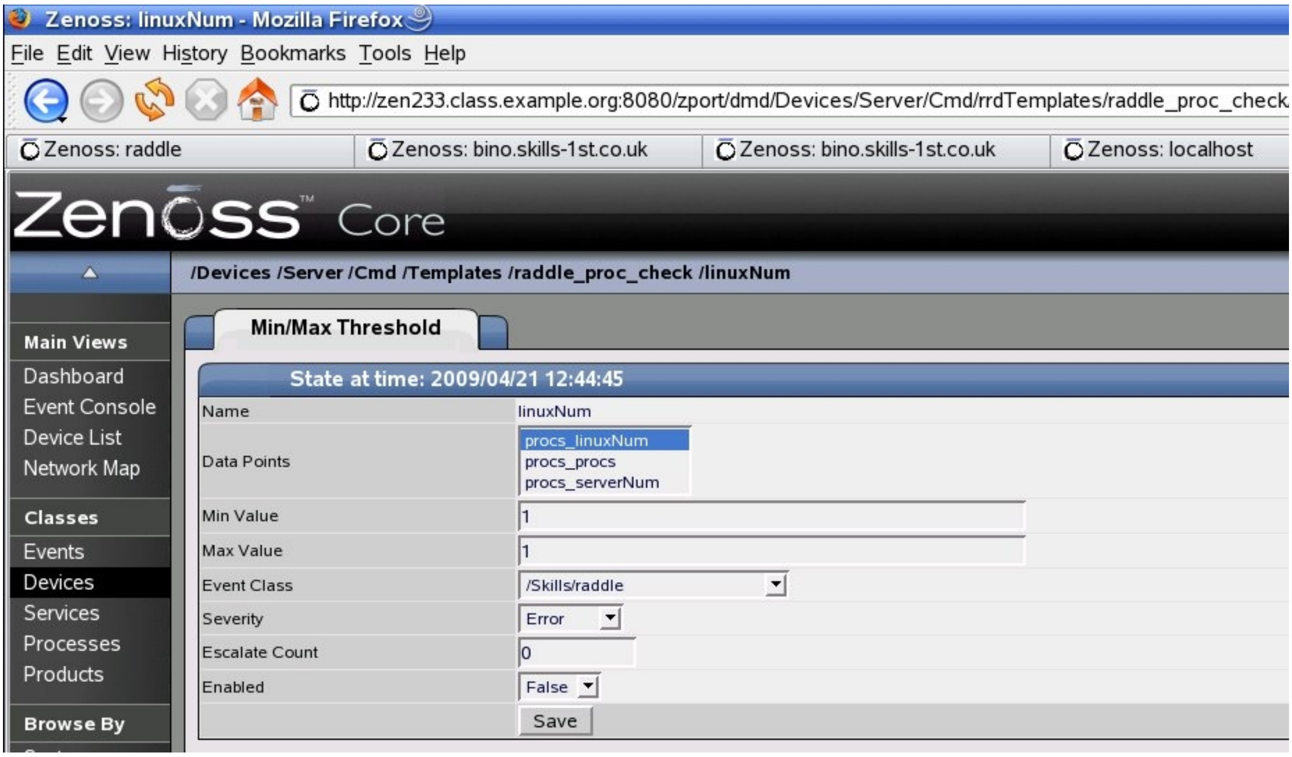
Name	Type	Data Points	Severity	Enabled
□ linuxNum	MinMaxThreshold	procs_linuxNum	Error	False
□ procs	MinMaxThreshold	procs_procs	Critical	False
□ serverNum	MinMaxThreshold	procs_serverNum	Warning	False

Graph Definitions

Seq	Name	Graph Points	Units	Height	Width
0	□ raddle_procs	linuxNum2, serverNum2, procs2, linuxNum, procs, serverNum		100	500

Figure 28: raddle_proc_check performance data collector template

As can be seen in the following screenshot, thresholds are chosen based on the defined DataPoints. Events of a specified class, of a given severity can be generated when the threshold is exceeded.


Zenoss: linuxNum - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Devices/Server/Cmd/rrdTemplates/raddle_proc_check
Zenoss: raddle
Zenoss: bino.skills-1st.co.uk
Zenoss: bino.skills-1st.co.uk
Zenoss: localhost
Zenoss Core
Main Views
Dashboard
Event Console
Device List
Network Map
Classes
Events
Devices
Services
Processes
Products
Browse By
Min/Max Threshold

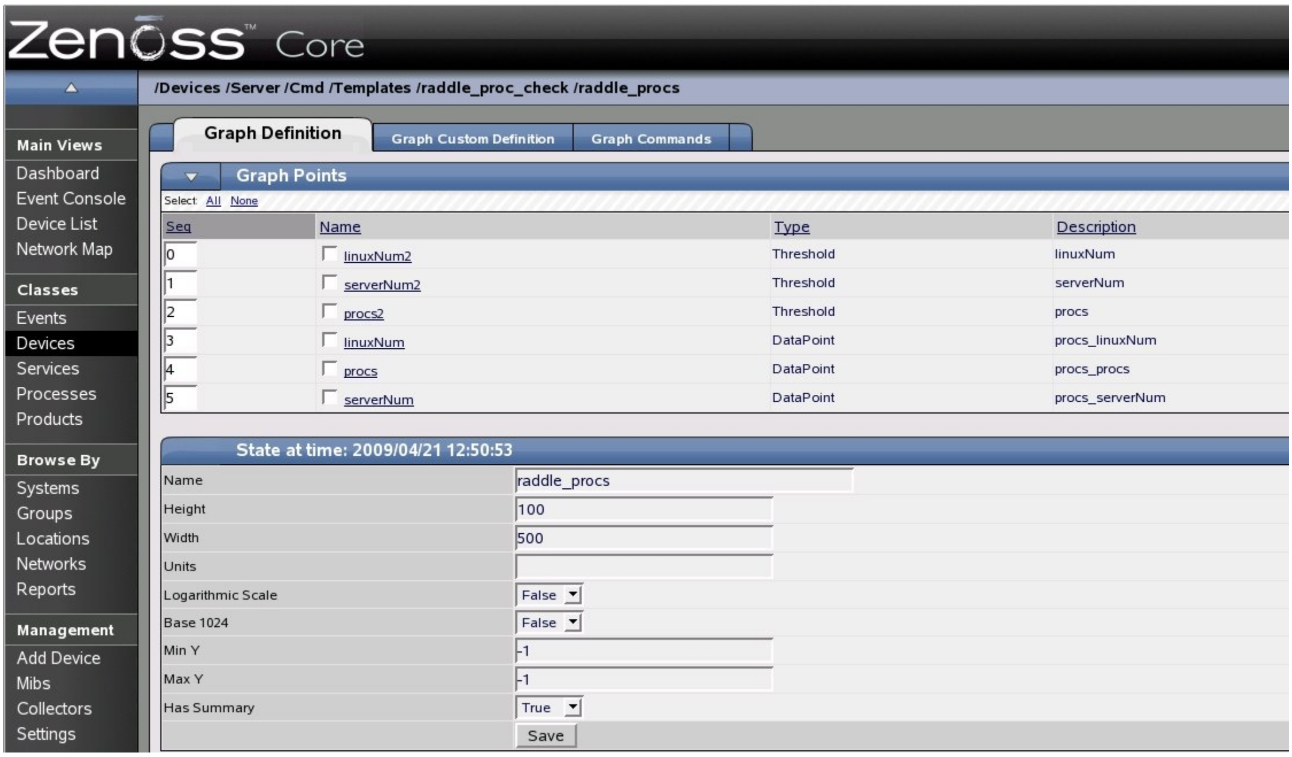
/Devices /Server /Cmd /Templates /raddle_proc_check /linuxNum

State at time: 2009/04/21 12:44:45

Name	linuxNum
Data Points	procs_linuxNum procs_procs procs_serverNum
Min Value	1
Max Value	1
Event Class	/Skills/raddle
Severity	Error
Escalate Count	0
Enabled	False
	Save

Figure 29: Defining a threshold for the procs_linuxNum DataPoint

As many graphs as are desired can be created. In this example, a single graph with all three DataPoints will be defined, including the three thresholds.



/Devices /Server /Cmd /Templates /raddle_proc_check /raddle_procs

Graph Points

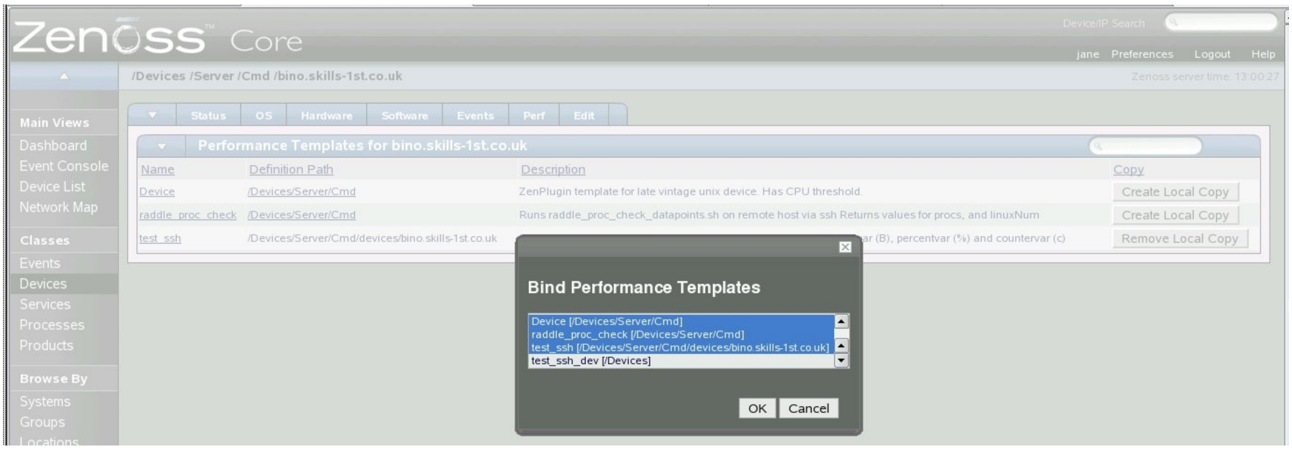
Seq	Name	Type	Description
0	□ linuxNum2	Threshold	linuxNum
1	□ serverNum2	Threshold	serverNum
2	□ procs2	Threshold	procs
3	□ linuxNum	DataPoint	procs_linuxNum
4	□ procs	DataPoint	procs_procs
5	□ serverNum	DataPoint	procs_serverNum

State at time: 2009/04/21 12:50:53

Name	raddle_procs
Height	100
Width	500
Units	
Logarithmic Scale	False
Base 1024	False
Min Y	-1
Max Y	-1
Has Summary	True

Figure 30: raddle_procs Graph Definition to plot DataPoints and Thresholds

This performance data collector template was defined for the class of devices */Server/Cmd*. To ensure that the template is applied to the host bino.skills-1st.co.uk, use the *More* -> *Templates* drop-down menu from the device's main page. From there, select the drop-down and *Bind Templates* menu. A popup box allows you to select templates to bind. **Note** that you should select **all** templates that you want bound (use Ctrl key to select multiple options) – just selecting the new template will de-select any templates already bound.



/Devices /Server /Cmd /bino.skills-1st.co.uk

Zenoss server time: 13:00:27

Performance Templates for bino.skills-1st.co.uk

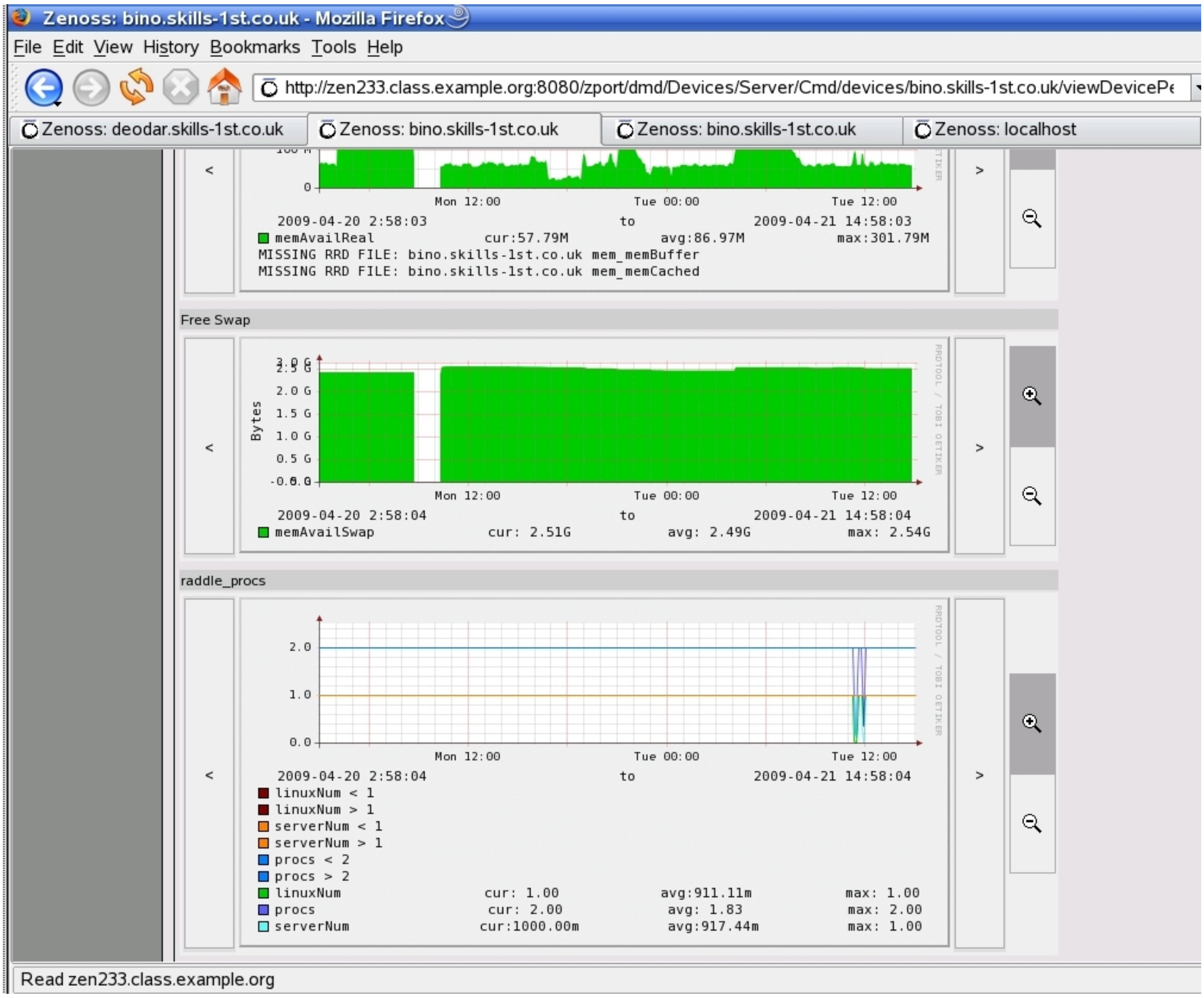
Name	Definition Path	Description	Copy
Device	/Devices/Server/Cmd	ZenPlugin template for late vintage unix device. Has CPU threshold.	Create Local Copy
raddle_proc_check	/Devices/Server/Cmd	Runs raddle_proc_check_datapoints.sh on remote host via ssh Returns values for procs, and linuxNum	Create Local Copy
test_ssh	/Devices/Server/Cmd/devices/bino.skills-1st.co.uk	ar (B), percentvar (%) and countervar (c)	Remove Local Copy

Bind Performance Templates

Device [/Devices/Server/Cmd]
raddle_proc_check [/Devices/Server/Cmd]
test_ssh [/Devices/Server/Cmd/devices/bino.skills-1st.co.uk]
test_ssh_dev [/Devices]

Figure 31: Binding multiple performance data collection templates to a device

Once the template is bound to a device or class of devices, data will start to appear under the *Performance* tab of a device.


Zenoss: bino.skills-1st.co.uk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Devices/Server/Cmd/devices/bino.skills-1st.co.uk/viewDevicePe
Zenoss: deodar.skills-1st.co.uk
Zenoss: bino.skills-1st.co.uk
Zenoss: bino.skills-1st.co.uk
Zenoss: localhost
0
Mon 12:00
Tue 00:00
Tue 12:00
2009-04-20 2:58:03
to
2009-04-21 14:58:03
🟩 memAvailReal cur:57.79M avg:86.97M max:301.79M
Free Swap
Bytes
2.0 G
1.5 G
1.0 G
0.5 G
-0.0
Mon 12:00
Tue 00:00
Tue 12:00
2009-04-20 2:58:04
to
2009-04-21 14:58:04
🟩 memAvailSwap cur: 2.51G avg: 2.49G max: 2.54G
raddle_procs
2.0
1.0
0.0
Mon 12:00
Tue 00:00
Tue 12:00
2009-04-20 2:58:04
to
2009-04-21 14:58:04
🟥 linuxNum < 1
🟥 linuxNum > 1
🟧 serverNum < 1
🟧 serverNum > 1
🟦 procs < 2
🟦 procs > 2
🟩 linuxNum cur: 1.00 avg:911.11m max: 1.00
🟪 procs cur: 2.00 avg: 1.83 max: 2.00
🟦 serverNum cur:1000.00m avg:917.44m max: 1.00
RRDTOOL / TOBI OETIKER

```
MISSING RRD FILE: bino.skills-1st.co.uk mem_memBuffer
MISSING RRD FILE: bino.skills-1st.co.uk mem_memCached
```

Read zen233.class.example.org

Figure 32: Performance graph for raddle_procs template (thresholds disabled)

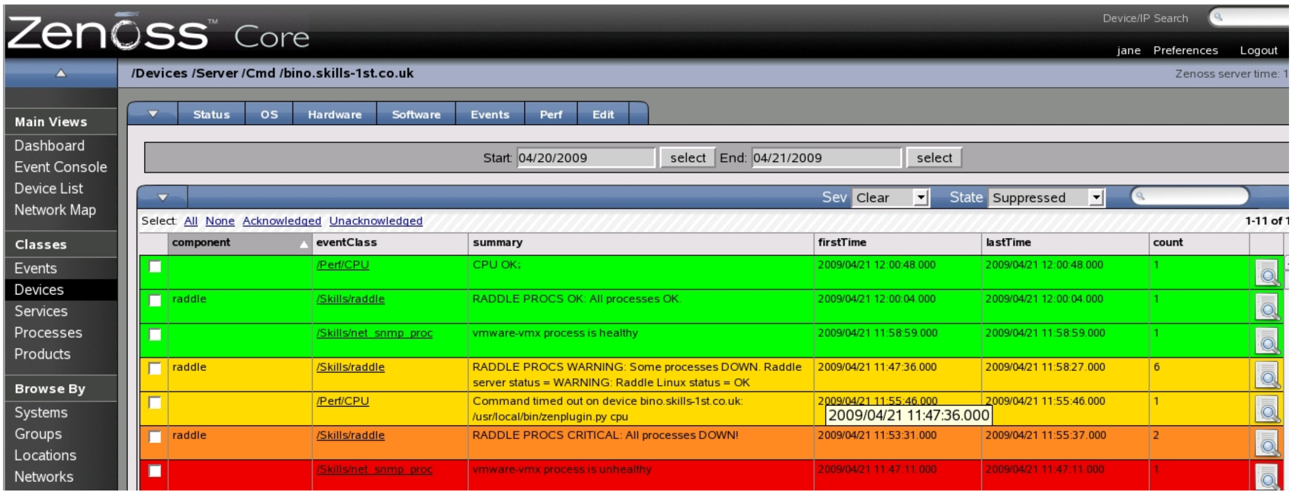
Note in Figure 32 above that thresholds have been disabled in the raddle_procs template, hence no threshold values are shown.

With command-driven performance data collectors, there are two opportunities for generating events:

- Using thresholds on DataPoints as described above
- Using the exit status from the script

If a script returns an exit status as defined by the Nagios plugin API, then events are automatically generated with a severity corresponding to the exit code:

- Script exit code of OK (0) Zenoss event severity = Clear (0)
- Script exit code of WARNING (1) Zenoss event severity = Warning (3)
- Script exit code of CRITICAL (2) Zenoss event severity = Error (4)



/Devices /Server /Cmd /bino.skills-1st.co.uk

Start: 04/20/2009 End: 04/21/2009

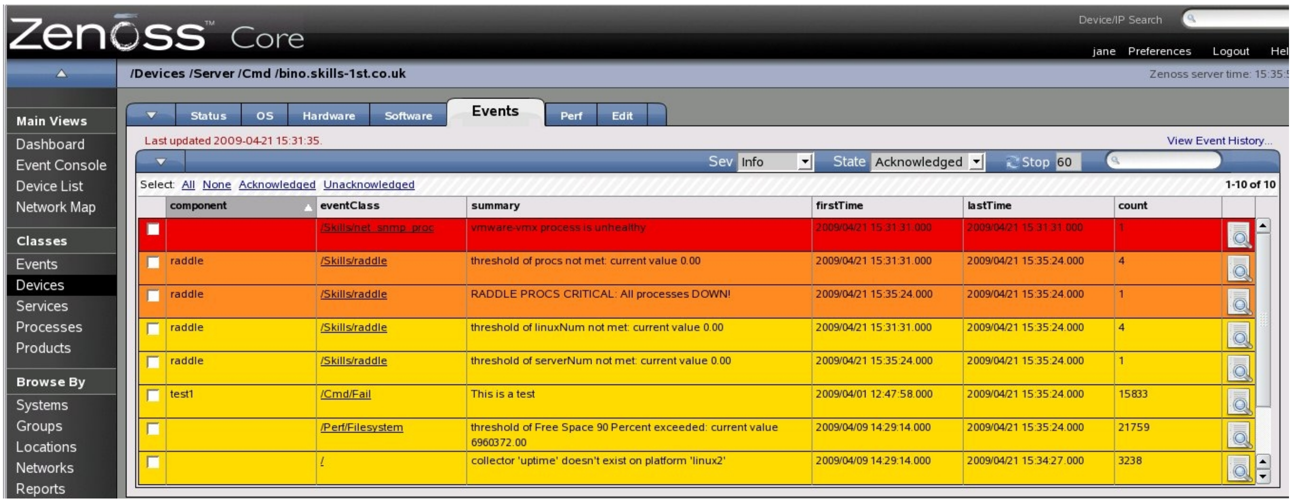
Sev Clear State Suppressed

component	eventClass	summary	firstTime	lastTime	count
	/Perf/CPU	CPU OK;	2009/04/21 12:00:48.000	2009/04/21 12:00:48.000	1
raddle	/Skills/raddle	RADDLE PROCS OK: All processes OK.	2009/04/21 12:00:04.000	2009/04/21 12:00:04.000	1
	/Skills/net_snmp_proc	vmware-vmx process is healthy	2009/04/21 11:58:59.000	2009/04/21 11:58:59.000	1
raddle	/Skills/raddle	RADDLE PROCS WARNING: Some processes DOWN. Raddle server status = WARNING: Raddle Linux status = OK	2009/04/21 11:47:36.000	2009/04/21 11:58:27.000	6
	/Perf/CPU	Command timed out on device bino.skills-1st.co.uk: /usr/local/bin/zenplugin.py cpu	2009/04/21 11:55:46.000	2009/04/21 11:55:46.000	1
raddle	/Skills/raddle	RADDLE PROCS CRITICAL: All processes DOWN!	2009/04/21 11:53:31.000	2009/04/21 11:55:37.000	2
	/Skills/net_snmp_proc	vmware-vmx process is unhealthy	2009/04/21 11:47:11.000	2009/04/21 11:47:11.000	1

Figure 33: Event console showing events generated by script Data Source

Note that the *eventClass* and the *component* fields of the event have been populated by the Data Source configuration. The “good news” event automatically clears the “bad news” events using Zenoss's default event correlation.

If the template thresholds are enabled then extra events are received, with their configured severities.



/Devices /Server /Cmd /bino.skills-1st.co.uk

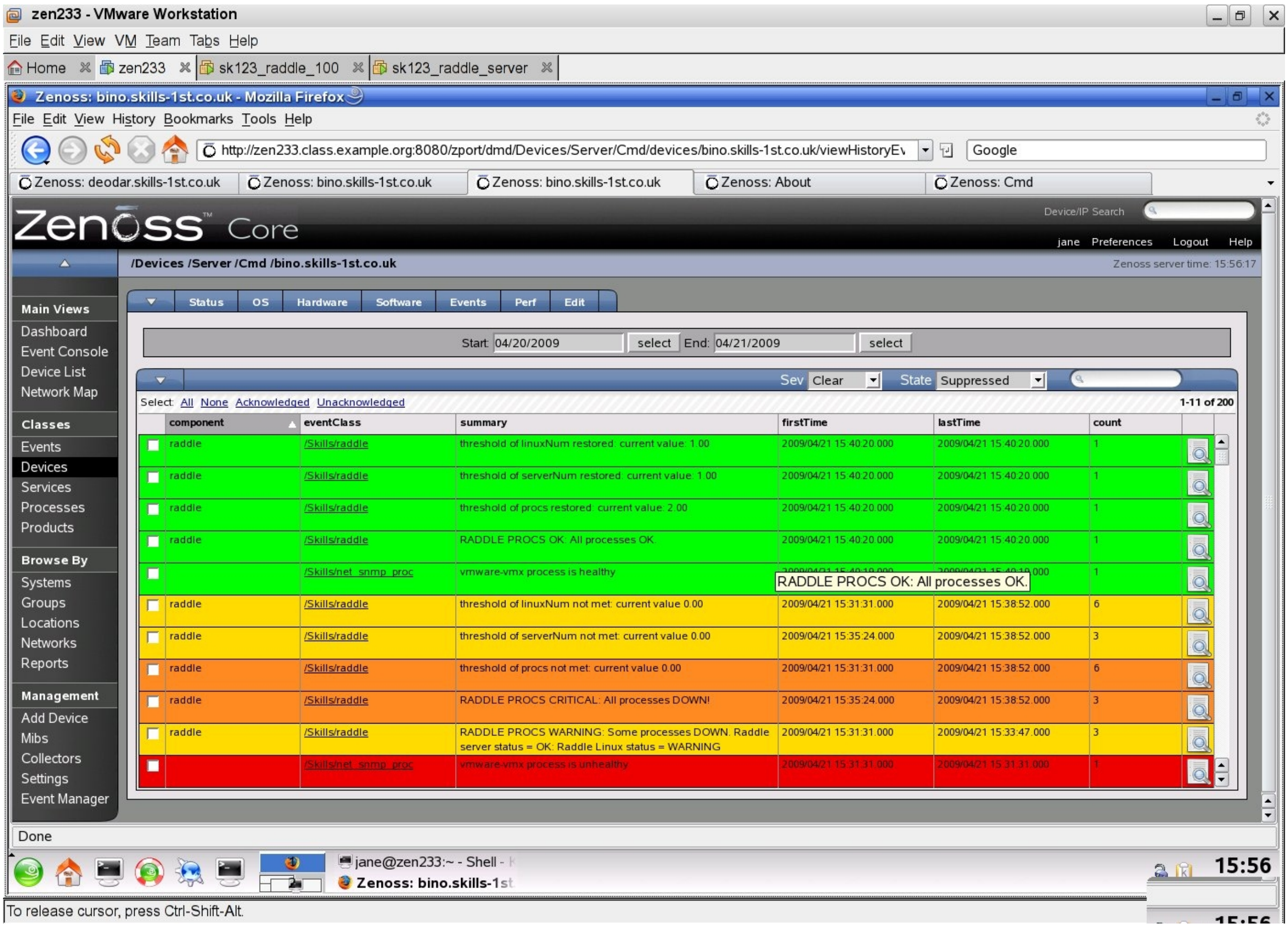
Last updated 2009-04-21 15:31:35.

Sev Info State Acknowledged Stop 60

component	eventClass	summary	firstTime	lastTime	count
	/Skills/net_snmp_proc	vmware-vmx process is unhealthy	2009/04/21 15:31:31.000	2009/04/21 15:31:31.000	1
raddle	/Skills/raddle	threshold of procs not met: current value 0.00	2009/04/21 15:31:31.000	2009/04/21 15:35:24.000	4
raddle	/Skills/raddle	RADDLE PROCS CRITICAL: All processes DOWN!	2009/04/21 15:35:24.000	2009/04/21 15:35:24.000	1
raddle	/Skills/raddle	threshold of linuxNum not met: current value 0.00	2009/04/21 15:31:31.000	2009/04/21 15:35:24.000	4
raddle	/Skills/raddle	threshold of serverNum not met: current value 0.00	2009/04/21 15:35:24.000	2009/04/21 15:35:24.000	1
test1	/Cmd/Fail	This is a test	2009/04/01 12:47:58.000	2009/04/21 15:35:24.000	15833
	/Perf/Filesystem	threshold of Free Space 90 Percent exceeded: current value 6960372.00	2009/04/09 14:29:14.000	2009/04/21 15:35:24.000	21759
	/	collector 'uptime' doesn't exist on platform 'linux2'	2009/04/09 14:29:14.000	2009/04/21 15:34:27.000	3238

Figure 34: Event console showing events generated by script data source and thresholds

Again, threshold “good news” events automatically clear “bad news”.



http://zen233.class.example.org:8080/zport/dmd/Devices/Server/Cmd/devices/bino.skills-1st.co.uk/viewHistoryEv

/Devices /Server /Cmd /bino.skills-1st.co.uk

Zenoss server time: 15:56:17

Start: 04/20/2009 End: 04/21/2009

Sev: Clear State: Suppressed

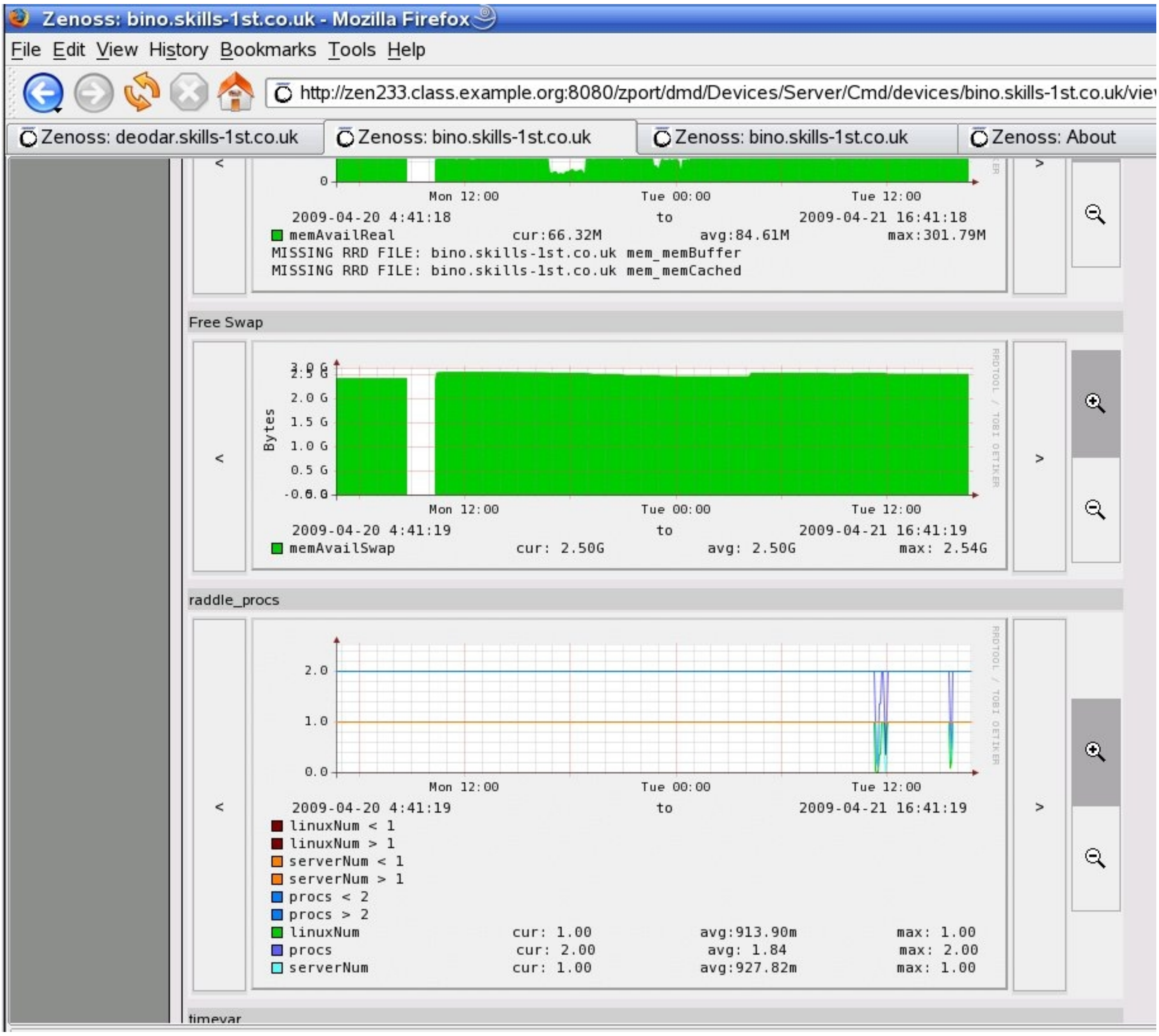
1-11 of 200

component	eventClass	summary	firstTime	lastTime	count
raddle	/Skills/raddle	threshold of linuxNum restored: current value: 1.00	2009/04/21 15:40:20.000	2009/04/21 15:40:20.000	1
raddle	/Skills/raddle	threshold of serverNum restored: current value: 1.00	2009/04/21 15:40:20.000	2009/04/21 15:40:20.000	1
raddle	/Skills/raddle	threshold of procs restored: current value: 2.00	2009/04/21 15:40:20.000	2009/04/21 15:40:20.000	1
raddle	/Skills/raddle	RADDLE PROCS OK: All processes OK.	2009/04/21 15:40:20.000	2009/04/21 15:40:20.000	1
	/Skills/net_snmp_proc	vmware-vmx process is healthy	2009/04/21 15:40:19.000	2009/04/21 15:40:19.000	1
raddle	/Skills/raddle	threshold of linuxNum not met: current value 0.00	2009/04/21 15:31:31.000	2009/04/21 15:38:52.000	6
raddle	/Skills/raddle	threshold of serverNum not met: current value 0.00	2009/04/21 15:35:24.000	2009/04/21 15:38:52.000	3
raddle	/Skills/raddle	threshold of procs not met: current value 0.00	2009/04/21 15:31:31.000	2009/04/21 15:38:52.000	6
raddle	/Skills/raddle	RADDLE PROCS CRITICAL: All processes DOWN!	2009/04/21 15:35:24.000	2009/04/21 15:38:52.000	3
raddle	/Skills/raddle	RADDLE PROCS WARNING: Some processes DOWN. Raddle server status = OK: Raddle Linux status = WARNING	2009/04/21 15:31:31.000	2009/04/21 15:33:47.000	3
	/Skills/net_snmp_proc	vmware-vmx process is unhealthy	2009/04/21 15:31:31.000	2009/04/21 15:31:31.000	1

RADDLE PROCS OK: All processes OK.

Figure 35: Event history showing "good news " and "bad news" events from scripts and thresholds

Threshold values are also shown on the performance graphs.


Zenoss: bino.skills-1st.co.uk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://zen233.class.example.org:8080/zport/dmd/Devices/Server/Cmd/devices/bino.skills-1st.co.uk/vie
Zenoss: deodar.skills-1st.co.uk
Zenoss: bino.skills-1st.co.uk
Zenoss: bino.skills-1st.co.uk
Zenoss: About
Free Swap
raddle_procs
timevar
Bytes
3.0 G
2.5 G
2.0 G
1.5 G
1.0 G
0.5 G
0.0
2.0
1.0
0.0
Mon 12:00
Tue 00:00
Tue 12:00
RRDTOOL / TOBI OETIKER

```
2009-04-20 4:41:18              to              2009-04-21 16:41:18
🟩 memAvailReal          cur:66.32M        avg:84.61M        max:301.79M
MISSING RRD FILE: bino.skills-1st.co.uk mem_memBuffer
MISSING RRD FILE: bino.skills-1st.co.uk mem_memCached
```

```
2009-04-20 4:41:19              to              2009-04-21 16:41:19
🟩 memAvailSwap         cur: 2.50G        avg: 2.50G        max: 2.54G
```

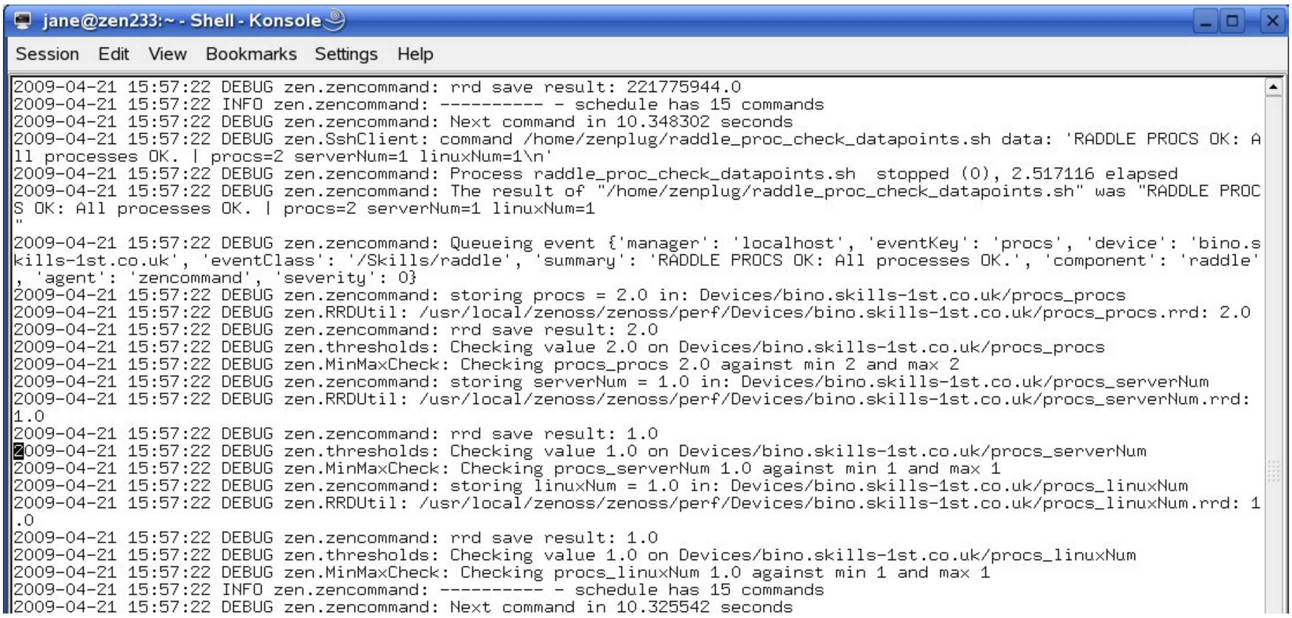
```
2009-04-20 4:41:19              to              2009-04-21 16:41:19
■ linuxNum < 1
■ linuxNum > 1
■ serverNum < 1
■ serverNum > 1
■ procs < 2
■ procs > 2
🟩 linuxNum             cur: 1.00        avg:913.90m        max: 1.00
■ procs                cur: 2.00        avg: 1.84          max: 2.00
■ serverNum            cur: 1.00        avg:927.82m        max: 1.00
```

Figure 36: Performance graphs for the raddle_procs template demonstrating enabled thresholds

To better understand how zencommand runs scripts and to help debugging, modify the parameters for zencommand to increase debugging in the logfile *$ZENHOME/log/zencommand.log*. Set:

```
logseverity 10
```

and recycle the zencommand daemon. This configuration can either be modified in the GUI from *Settings -> Daemons* and use the *edit config* link and the *Restart* button; alternatively edit *$ZENHOME/etc/zencommand.conf* directly and then restart zencommand with *zencommand restart* (you will need to be the zenoss user).



```
2009-04-21 15:57:22 DEBUG zen.zencommand: rrd save result: 221775944.0
2009-04-21 15:57:22 INFO zen.zencommand: ---------- - schedule has 15 commands
2009-04-21 15:57:22 DEBUG zen.zencommand: Next command in 10.348302 seconds
2009-04-21 15:57:22 DEBUG zen.SshClient: command /home/zenplug/raddle_proc_check_datapoints.sh data: 'RADDLE PROCS OK: A
ll processes OK. | procs=2 serverNum=1 linuxNum=1\n'
2009-04-21 15:57:22 DEBUG zen.zencommand: Process raddle_proc_check_datapoints.sh  stopped (0), 2.517116 elapsed
2009-04-21 15:57:22 DEBUG zen.zencommand: The result of "/home/zenplug/raddle_proc_check_datapoints.sh" was "RADDLE PROC
S OK: All processes OK. | procs=2 serverNum=1 linuxNum=1
"
2009-04-21 15:57:22 DEBUG zen.zencommand: Queueing event {'manager': 'localhost', 'eventKey': 'procs', 'device': 'bino.s
kills-1st.co.uk', 'eventClass': '/Skills/raddle', 'summary': 'RADDLE PROCS OK: All processes OK.', 'component': 'raddle'
, 'agent': 'zencommand', 'severity': 0}
2009-04-21 15:57:22 DEBUG zen.zencommand: storing procs = 2.0 in: Devices/bino.skills-1st.co.uk/procs_procs
2009-04-21 15:57:22 DEBUG zen.RRDUtil: /usr/local/zenoss/zenoss/perf/Devices/bino.skills-1st.co.uk/procs_procs.rrd: 2.0
2009-04-21 15:57:22 DEBUG zen.zencommand: rrd save result: 2.0
2009-04-21 15:57:22 DEBUG zen.thresholds: Checking value 2.0 on Devices/bino.skills-1st.co.uk/procs_procs
2009-04-21 15:57:22 DEBUG zen.MinMaxCheck: Checking procs_procs 2.0 against min 2 and max 2
2009-04-21 15:57:22 DEBUG zen.zencommand: storing serverNum = 1.0 in: Devices/bino.skills-1st.co.uk/procs_serverNum
2009-04-21 15:57:22 DEBUG zen.RRDUtil: /usr/local/zenoss/zenoss/perf/Devices/bino.skills-1st.co.uk/procs_serverNum.rrd:
1.0
2009-04-21 15:57:22 DEBUG zen.zencommand: rrd save result: 1.0
2009-04-21 15:57:22 DEBUG zen.thresholds: Checking value 1.0 on Devices/bino.skills-1st.co.uk/procs_serverNum
2009-04-21 15:57:22 DEBUG zen.MinMaxCheck: Checking procs_serverNum 1.0 against min 1 and max 1
2009-04-21 15:57:22 DEBUG zen.zencommand: storing linuxNum = 1.0 in: Devices/bino.skills-1st.co.uk/procs_linuxNum
2009-04-21 15:57:22 DEBUG zen.RRDUtil: /usr/local/zenoss/zenoss/perf/Devices/bino.skills-1st.co.uk/procs_linuxNum.rrd: 1
.0
2009-04-21 15:57:22 DEBUG zen.zencommand: rrd save result: 1.0
2009-04-21 15:57:22 DEBUG zen.thresholds: Checking value 1.0 on Devices/bino.skills-1st.co.uk/procs_linuxNum
2009-04-21 15:57:22 DEBUG zen.MinMaxCheck: Checking procs_linuxNum 1.0 against min 1 and max 1
2009-04-21 15:57:22 INFO zen.zencommand: ---------- - schedule has 15 commands
2009-04-21 15:57:22 DEBUG zen.zencommand: Next command in 10.325542 seconds
```

Figure 37: Fragment of $ZENHOME/log/zencommand.log showing raddle_proc_check_datapoints.sh

The zencommand.log shows:

- The remote script being run by zen.SshClient, including the returned output
- zen.zencommand queueing an event, including the configured eventClass, component and with the event summary field set to the text information output (everything before the vertical bar in the script output line). The eventKey field is set to the Data Source name.
- zen.RRDUtil storing away the latest values
- zen.thresholds and zen.MinMaxCheck checking the latest values against the configured thresholds

5.2.2 Using Zenoss to run Nagios plugins through SSH

Nagios plugins integrate with Zenoss in a very similar manner to running standalone commands. Nagios plugins will automatically be installed on a Zenoss manger under */usr/local/zenoss/common/libexec*. Some Nagios plugins can be used to check details of remote systems, for example the check_http plugin tests URLs on a given destination system:

```
        check_http -H www.skills-1st.co.uk
```

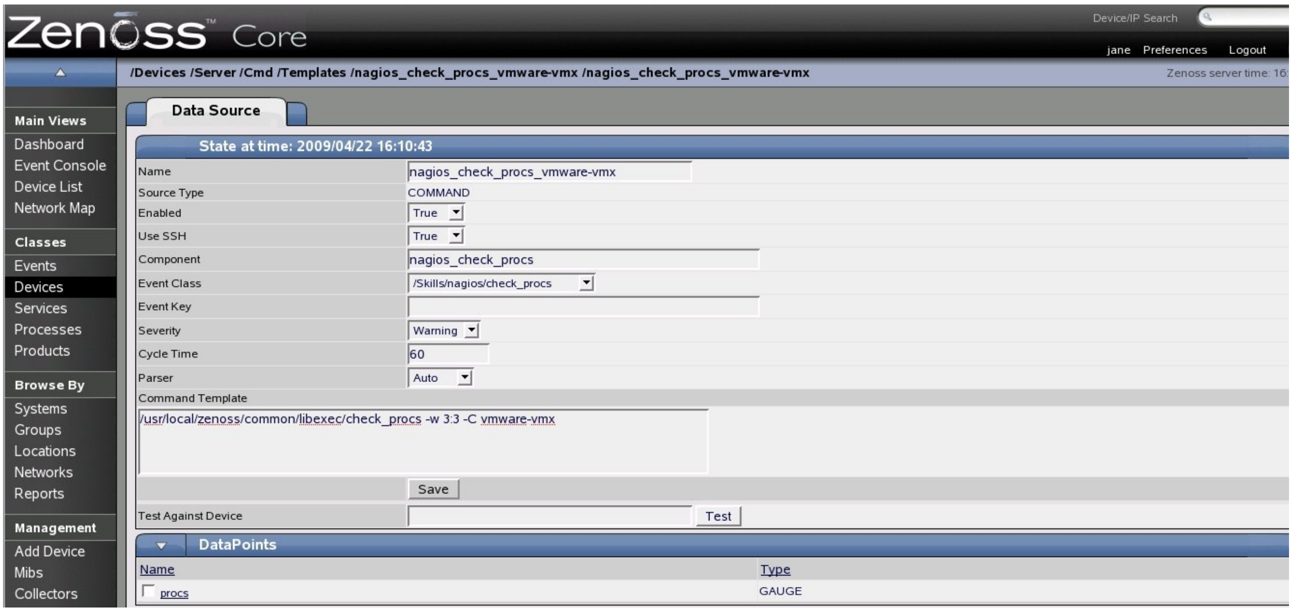
Some Nagios plugins are designed to check details on a **local** system, such as the check_procs plugin.

It is perfectly possible to install the check_procs Nagios plugin standalone on a remote system and it can be placed in any directory. As an example, install check_procs into /usr/local/zenoss/common/libexec on a remote system (not a Zenoss manager). Ensure that the plugin runs standalone, locally, by:

```
cd /usr/local/zenoss/common/libexec
./check_procs -w 1:4 -c 1:10 -C sshd
```

Next ensure that the zProperties for this device are setup in the Zenoss GUI to permit SSH communications between the Zenoss manager and the remote device. This is exactly the same as described in Figure 24 above for running standalone SSH commands.

To utilise information from the Nagios plugin, setup a Zenoss performance data collection template in the same way as described above.



/Devices /Server /Cmd /Templates /nagios_check_procs_vmware-vmx /nagios_check_procs_vmware-vmx

State at time: 2009/04/22 16:10:43

Name	nagios_check_procs_vmware-vmx
Source Type	COMMAND
Enabled	True
Use SSH	True
Component	nagios_check_procs
Event Class	/Skills/nagios/check_procs
Event Key	
Severity	Warning
Cycle Time	60
Parser	Auto

Command Template

```
/usr/local/zenoss/common/libexec/check_procs -w 3:3 -C vmware-vmx
```

Test Against Device

DataPoints

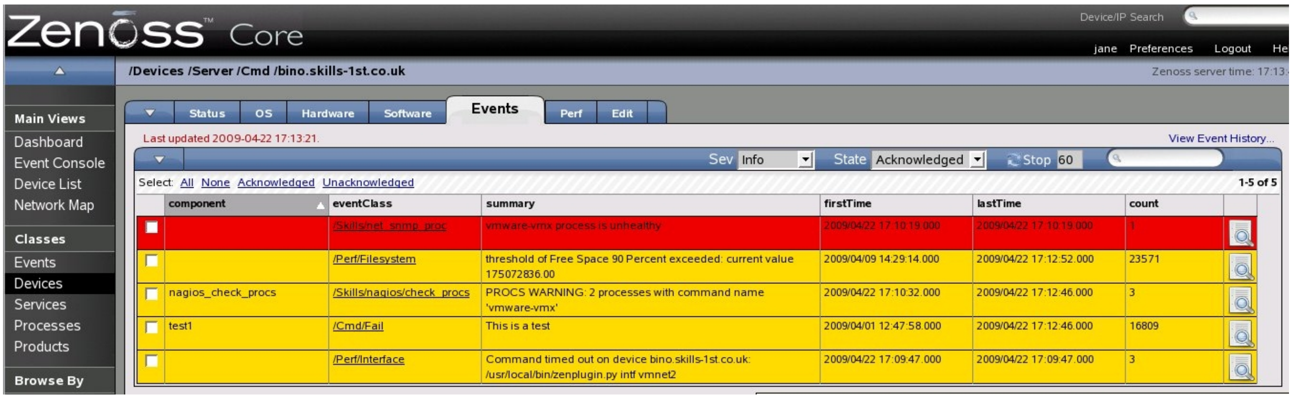
Name	Type
□ procs	GAUGE

Figure 38: Performance data collection template using SSH to run remote Nagios check_procs plugin

Note that in this case, the full path to the plugin is supplied. It is checking for exactly 3 occurrences of a short process name *vmware-vmx*. The component field is set to *nagios_check_procs* and a new event class of */Skills/nagios/check_procs* has been created for use with this template.

The advantage of using Zenoss plugins is that there are lots available in the community. The disadvantage is that many of them do **not** provide performance data values, simply a status and informational text. This means that creating DataPoints in Zenoss from which to create thresholds and graphs is not useful; although DataPoints **can** be specified, they have to exactly match the label of the data delivered by the plugin (which doesn't exist), so any graphs based on such DataPoints will have no data.

This doesn't mean that the Nagios check_procs plugin is necessarily useless. The plugin can specify warning and critical ranges for metrics (such as number of instances of a process, memory used, percentage CPU used) and delivers an exit status from the script which will drive Zenoss events.



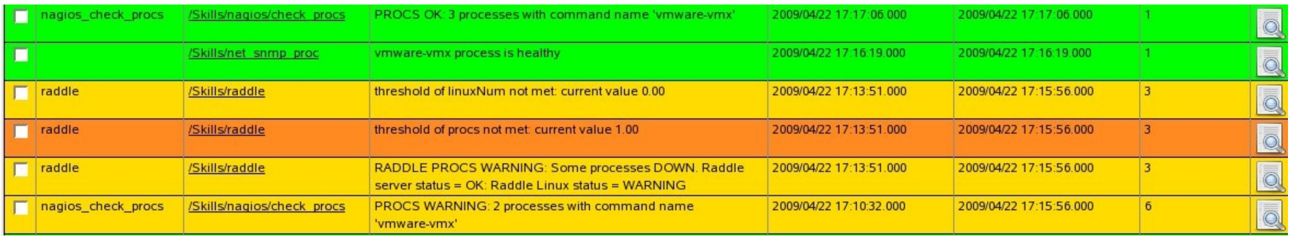
Last updated 2009-04-22 17:13:21.

Sev Info State Acknowledged Stop 60

	component	eventClass	summary	firstTime	lastTime	count	
□		/Skills/net_snmp_proc	vmware-vmx process is unhealthy	2009/04/22 17:10:19.000	2009/04/22 17:10:19.000	1	
□		/Perf/Filesystem	threshold of Free Space 90 Percent exceeded: current value 175072836.00	2009/04/09 14:29:14.000	2009/04/22 17:12:52.000	23571	
□	nagios_check_procs	/Skills/nagios/check_procs	PROCS WARNING: 2 processes with command name 'vmware-vmx'	2009/04/22 17:10:32.000	2009/04/22 17:12:46.000	3	
□	test1	/Cmd/Fail	This is a test	2009/04/01 12:47:58.000	2009/04/22 17:12:46.000	16809	
□		/Perf/Interface	Command timed out on device bino.skills-1st.co.uk: /usr/local/bin/zenplugin.py intf vmnet2	2009/04/22 17:09:47.000	2009/04/22 17:09:47.000	3	

Figure 39: Event console with warning event generated by Nagios check_procs plugin

As discussed with standalone events, the Nagios plugin "good news" status will deliver a Zenoss event with Cleared status; thus Nagios-driven "good news" events will automatically close their corresponding "bad news" events.



□	nagios_check_procs	/Skills/nagios/check_procs	PROCS OK: 3 processes with command name 'vmware-vmx'	2009/04/22 17:17:06.000	2009/04/22 17:17:06.000	1	
□		/Skills/net_snmp_proc	vmware-vmx process is healthy	2009/04/22 17:16:19.000	2009/04/22 17:16:19.000	1	
□	raddle	/Skills/raddle	threshold of linuxNum not met: current value 0.00	2009/04/22 17:13:51.000	2009/04/22 17:15:56.000	3	
□	raddle	/Skills/raddle	threshold of procs not met: current value 1.00	2009/04/22 17:13:51.000	2009/04/22 17:15:56.000	3	
□	raddle	/Skills/raddle	RADDLE PROCS WARNING: Some processes DOWN. Raddle server status = OK: Raddle Linux status = WARNING	2009/04/22 17:13:51.000	2009/04/22 17:15:56.000	3	
□	nagios_check_procs	/Skills/nagios/check_procs	PROCS WARNING: 2 processes with command name 'vmware-vmx'	2009/04/22 17:10:32.000	2009/04/22 17:15:56.000	6	

Figure 40: Event history console with "good news" and "bad news" events generated by Nagios plugin

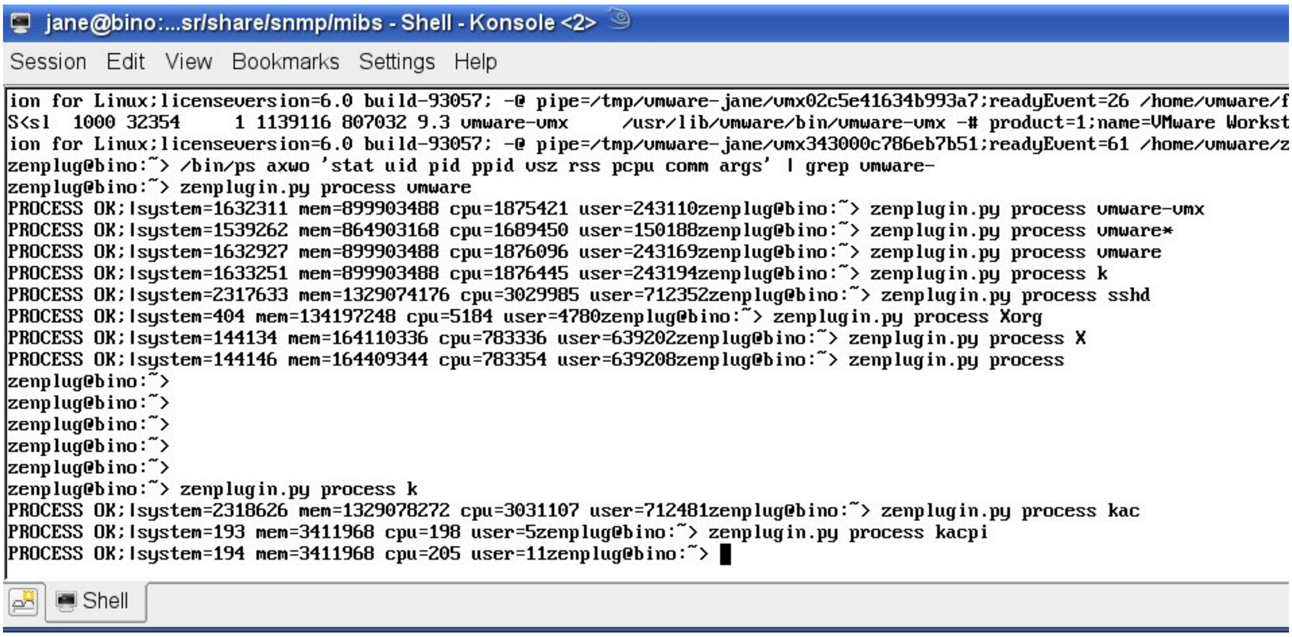
5.2.3 Using Zenoss to run Zenoss plugins through SSH

The Zenoss plugins are Python libraries run by the zenplugin.py command. The Zenoss plugins are not installed by default, even on the Zenoss manager, but they are easily downloaded and installed as described in section 3.3.

Documentation for the Zenoss plugins is rather light, especially around the *process* plugin; however the code can be examined, typically in:

```
/usr/local/lib/python2.5/site-packages/zenoss/plugins/linux2.py
```

This shows that a parameter is required to describe the process(es) to be monitored. This parameter will match any process that includes that string so processes can be specified as fully-qualified pathnames or short commands (try using *zenplugin.py process k* on a system that uses kde – it reports the totals of resources of all processes that include the letter k).



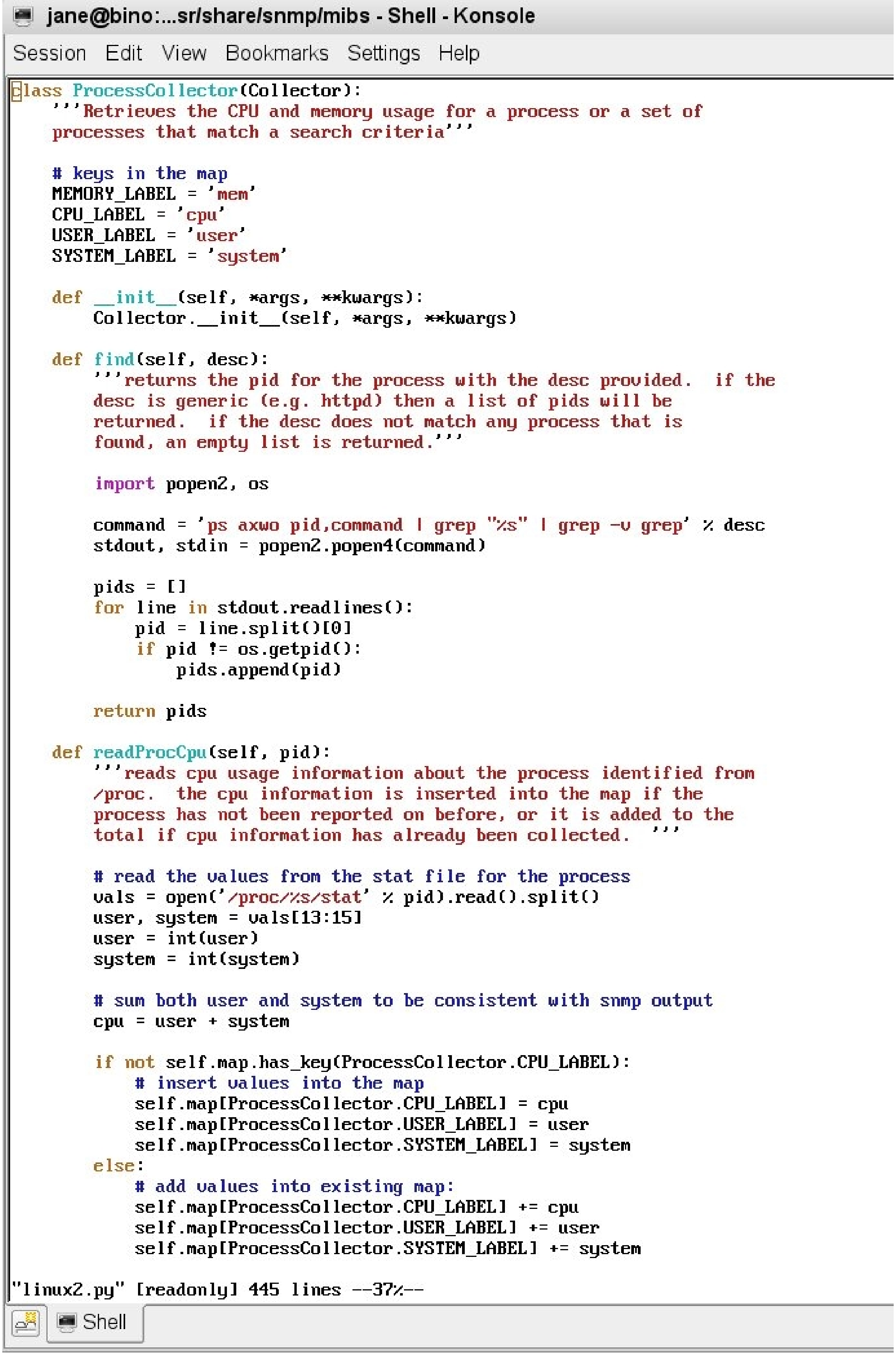
jane@bino:...sr/share/snmp/mibs - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
Shell

```
ion for Linux;licenseversion=6.0 build-93057; -@ pipe=/tmp/vmware-jane/vmx02c5e41634b993a7;readyEvent=26 /home/vmware/f
S<sl  1000 32354     1 1139116 807032 9.3 vmware-vmx     /usr/lib/vmware/bin/vmware-vmx -# product=1;name=VMware Workst
ion for Linux;licenseversion=6.0 build-93057; -@ pipe=/tmp/vmware-jane/vmx343000c786eb7b51;readyEvent=61 /home/vmware/z
zenplug@bino:~> /bin/ps axwo 'stat uid pid ppid vsz rss pcpu comm args' | grep vmware-
zenplug@bino:~> zenplugin.py process vmware
PROCESS OK;|system=1632311 mem=899903488 cpu=1875421 user=243110zenplug@bino:~> zenplugin.py process vmware-vmx
PROCESS OK;|system=1539262 mem=864903168 cpu=1689450 user=150188zenplug@bino:~> zenplugin.py process vmware*
PROCESS OK;|system=1632927 mem=899903488 cpu=1876096 user=243169zenplug@bino:~> zenplugin.py process vmware
PROCESS OK;|system=1633251 mem=899903488 cpu=1876445 user=243194zenplug@bino:~> zenplugin.py process k
PROCESS OK;|system=2317633 mem=1329074176 cpu=3029985 user=712352zenplug@bino:~> zenplugin.py process sshd
PROCESS OK;|system=404 mem=134197248 cpu=5184 user=4780zenplug@bino:~> zenplugin.py process Xorg
PROCESS OK;|system=144134 mem=164110336 cpu=783336 user=639202zenplug@bino:~> zenplugin.py process X
PROCESS OK;|system=144146 mem=164409344 cpu=783354 user=639208zenplug@bino:~> zenplugin.py process
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~>
zenplug@bino:~> zenplugin.py process k
PROCESS OK;|system=2318626 mem=1329078272 cpu=3031107 user=712481zenplug@bino:~> zenplugin.py process kac
PROCESS OK;|system=193 mem=3411968 cpu=198 user=5zenplug@bino:~> zenplugin.py process kacpi
PROCESS OK;|system=194 mem=3411968 cpu=205 user=11zenplug@bino:~> █
```

Figure 41: Invocations of zenplugin.py process with different process matching parameters

There appears to be no way to specify a way to count instances of a process. If there are multiple processes that match the description, then the CPU and memory values are summed for all matching processes.

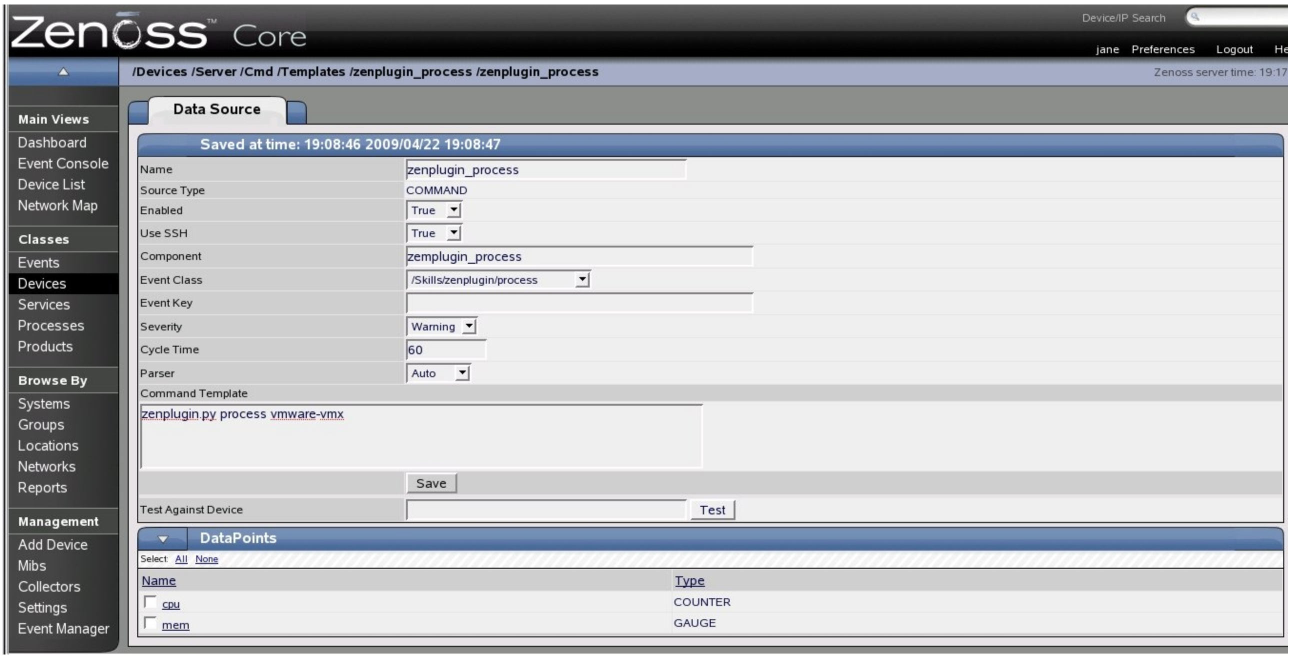
The plugin script shows that raw data is gathered by reading the stat file for the process in /proc/<process >. The “CPU” figure is derived by adding the “user” and “system” values and is reported in “jiffies” (1/100 second) that this process has been scheduled. The memory figure takes the resident set size of the process (plus 3 – for administrative purposes), and multiplies by pagesize to produce a memory figure in bytes.


jane@bino:...sr/share/snmp/mibs - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
class ProcessCollector(Collector):
    '''Retrieves the CPU and memory usage for a process or a set of
    processes that match a search criteria'''

    # keys in the map
    MEMORY_LABEL = 'mem'
    CPU_LABEL = 'cpu'
    USER_LABEL = 'user'
    SYSTEM_LABEL = 'system'

    def __init__(self, *args, **kwargs):
        Collector.__init__(self, *args, **kwargs)

    def find(self, desc):
        '''returns the pid for the process with the desc provided.  if the
        desc is generic (e.g. httpd) then a list of pids will be
        returned.  if the desc does not match any process that is
        found, an empty list is returned.'''

        import popen2, os

        command = 'ps axwo pid,command | grep "%s" | grep -v grep' % desc
        stdout, stdin = popen2.popen4(command)

        pids = []
        for line in stdout.readlines():
            pid = line.split()[0]
            if pid != os.getpid():
                pids.append(pid)

        return pids

    def readProcCpu(self, pid):
        '''reads cpu usage information about the process identified from
        /proc.  the cpu information is inserted into the map if the
        process has not been reported on before, or it is added to the
        total if cpu information has already been collected.  '''

        # read the values from the stat file for the process
        vals = open('/proc/%s/stat' % pid).read().split()
        user, system = vals[13:15]
        user = int(user)
        system = int(system)

        # sum both user and system to be consistent with snmp output
        cpu = user + system

        if not self.map.has_key(ProcessCollector.CPU_LABEL):
            # insert values into the map
            self.map[ProcessCollector.CPU_LABEL] = cpu
            self.map[ProcessCollector.USER_LABEL] = user
            self.map[ProcessCollector.SYSTEM_LABEL] = system
        else:
            # add values into existing map:
            self.map[ProcessCollector.CPU_LABEL] += cpu
            self.map[ProcessCollector.USER_LABEL] += user
            self.map[ProcessCollector.SYSTEM_LABEL] += system

"linux2.py" [readonly] 445 lines --37%--
```

Figure 42: Zenoss plugin linux2.py showing process collection code

Zenoss plugins can be used in exactly the same way as standalone scripts or Nagios plugins. Performance data collector templates can be created that call zenplugin.py on a remote system, using the SSH zProperties configured for a device.



/Devices /Server /Cmd /Templates /zenplugin_process /zenplugin_process

Saved at time: 19:08:46 2009/04/22 19:08:47

Name	zenplugin_process
Source Type	COMMAND
Enabled	True
Use SSH	True
Component	zemplugin_process
Event Class	/Skills/zenplugin/process
Event Key	
Severity	Warning
Cycle Time	60
Parser	Auto
Command Template	zenplugin.py process vmware-vmx
Test Against Device	

DataPoints

Name	Type
□ cpu	COUNTER
□ mem	GAUGE

Figure 43: Performance data collection template using Zenoss process plugin

In Figure 43 a new component value has been created, *zenplugin_process*, and a new event class is referenced (*/Skills/zenplugin/process*). Note that the Command Template field specifies a short name for *zenplugin.py*; this assumes that any device that has the template bound, will have the zCommandPath zProperty set to */usr/local/bin*.

The names of the DataPoints exactly match the label names of the CPU and mem output of the Zenoss plugin. Note that the CPU DataPoint has the COUNTER type; since CPU is the number of jiffies that the process has been scheduled, it will always be an increasing number, whereas mem can go up and down so the GAUGE type is more appropriate for mem. The COUNTER data type means that any graphs using it will automatically display rate-of-change, rather than the absolute value which is simply a large number that gradually increases.

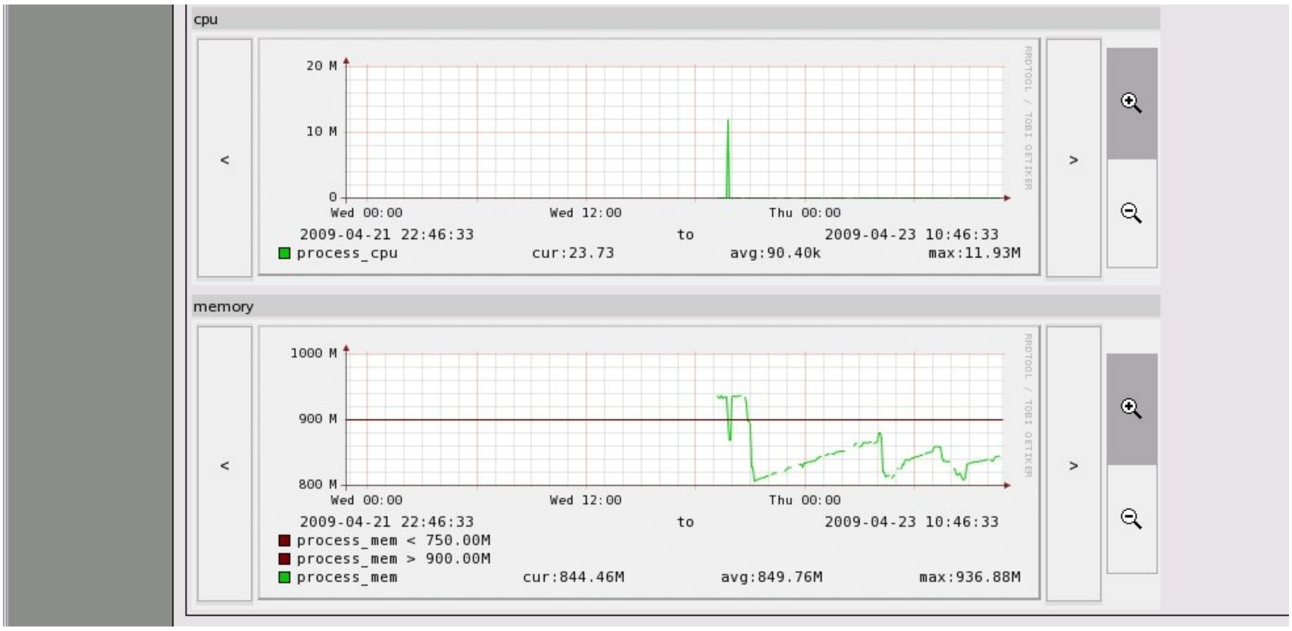
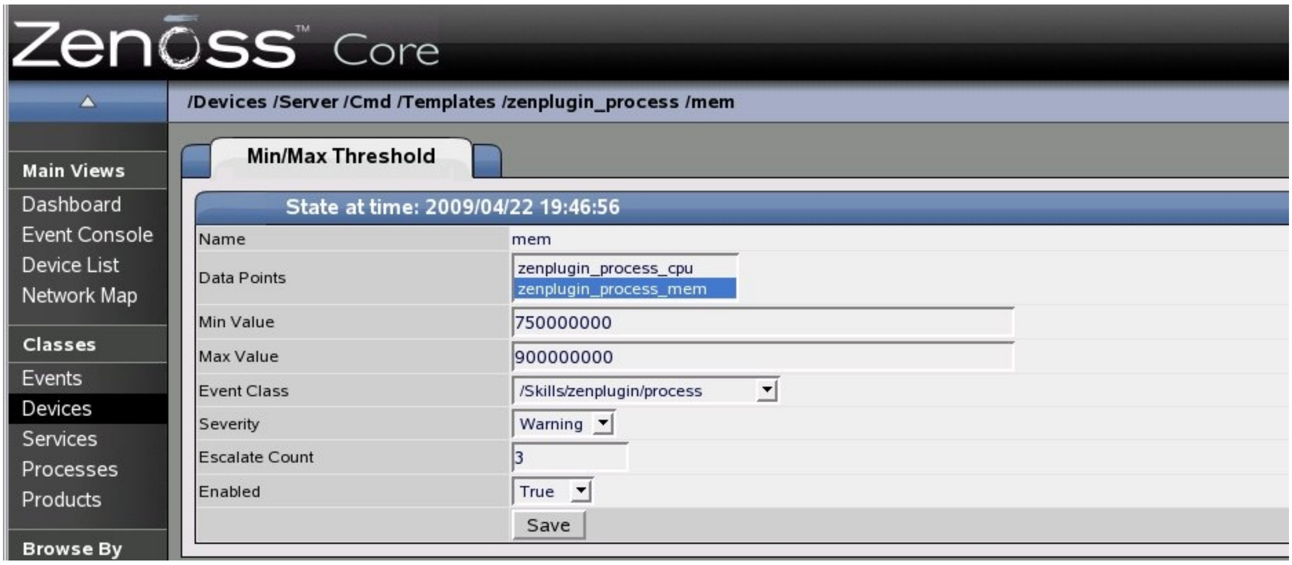


Figure 44: Performance graphs and thresholds for data gathered by the Zenoss process plugin

Zenoss plugins provide different benefits to the Nagios plugins. You cannot count instances of a process but, if you want the total CPU and memory resource used by the total number of invocations of a particular process, then the Zenoss process plugin matches that paradigm nicely. The other advantage of Zenoss plugins is that they not only deliver output in Nagios API format, but they also tend to deliver performance data in addition to the status and information text; hence they are more amenable to being used directly to supply data for graphs and thresholds (indeed, all the standard templates for */Server/Cmd* devices uses Zenoss plugins).

The negative side is that there is no way within the Zenoss process plugin to set acceptable thresholds for CPU and memory so the exit status is always “OK” unless the plugin itself had problems retrieving data.. This means that if events are required on thresholds based on the Zenoss plugin data, then thresholds must be setup within the Zenoss performance data collector template – there are no “automatic” events.



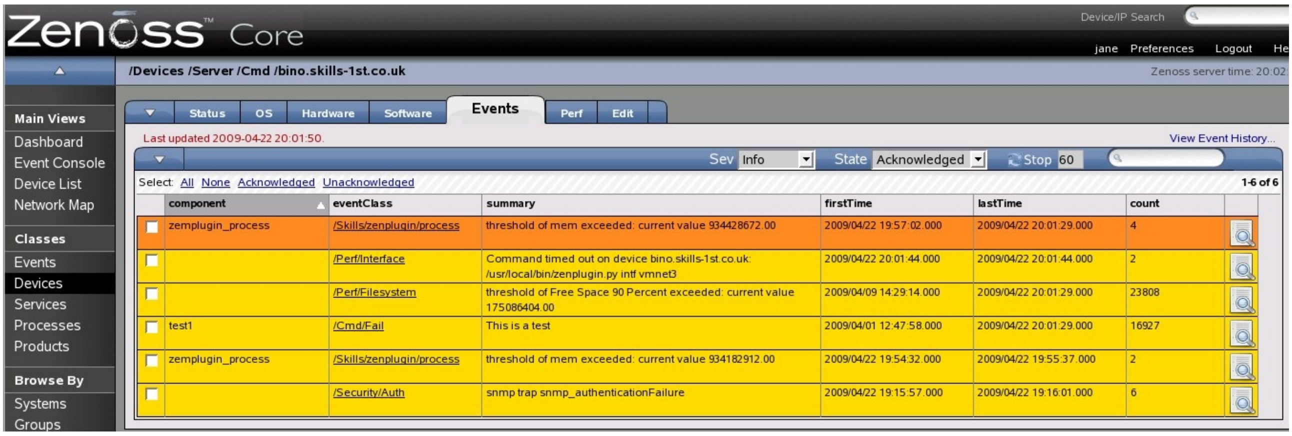
/Devices /Server /Cmd /Templates /zenplugin_process /mem

State at time: 2009/04/22 19:46:56

Name	mem
Data Points	zenplugin_process_cpu zenplugin_process_mem
Min Value	750000000
Max Value	900000000
Event Class	/Skills/zenplugin/process
Severity	Warning
Escalate Count	3
Enabled	True

Figure 45: Threshold on memory for Zenoss process plugin DataPoint

Note that the threshold shown above demonstrates the use of the *Escalate Count* field. When the third similar event arrives, the severity will be escalated from the configured *Warning* to the next level, *Error*.



/Devices /Server /Cmd /bino.skills-1st.co.uk

Zenoss server time: 20:02

Last updated 2009-04-22 20:01:50.

Sev Info State Acknowledged Stop 60

Select: All None Acknowledged Unacknowledged 1-6 of 6

	component	eventClass	summary	firstTime	lastTime	count
□	zemplugin_process	/Skills/zenplugin/process	threshold of mem exceeded: current value 934428672.00	2009/04/22 19:57:02.000	2009/04/22 20:01:29.000	4
□		/Perf/Interface	Command timed out on device bino.skills-1st.co.uk: /usr/local/bin/zenplugin.py intf vmnet3	2009/04/22 20:01:44.000	2009/04/22 20:01:44.000	2
□		/Perf/Filesystem	threshold of Free Space 90 Percent exceeded: current value 175086404.00	2009/04/09 14:29:14.000	2009/04/22 20:01:29.000	23808
□	test1	/Cmd/Fail	This is a test	2009/04/01 12:47:58.000	2009/04/22 20:01:29.000	16927
□	zemplugin_process	/Skills/zenplugin/process	threshold of mem exceeded: current value 934182912.00	2009/04/22 19:54:32.000	2009/04/22 19:55:37.000	2
□		/Security/Auth	snmp trap snmp_authenticationFailure	2009/04/22 19:15:57.000	2009/04/22 19:16:01.000	6

Figure 46: Event console showing /Skills/zenplugin/process threshold event escalated from Warning to Error

Events are generated by Zenoss when the threshold is exceeded and, as with all the other techniques already discussed, "good news" thresholds will automatically close "bad news" threshold events.

To summarise, the Zenoss plugins are better performance data collectors and the Nagios plugins more easily deliver threshold events.

6 Conclusions

A number of different process monitoring techniques have been discussed, each having their own merits. If devices cannot be monitored using SNMP, perhaps because of firewall limitations, then SSH provides access for standalone commands, Nagios plugins and Zenoss plugins. The choice between these three depends on what aspects of process monitoring are required.

Standalone scripts are the most flexible but you have to develop, test, maintain and deliver them.

Many Nagios plugins are available in the community but the standard check_procs offering does not deliver performance data and there is still the task of delivering the Nagios plugin to the remote system. check_procs does provide a flexible way for defining a “healthy” process and can automatically generate events based on this health.

Zenoss plugins also need installing remotely and add the prerequisite of a Python environment, but the Zenoss process plugin is good for delivering CPU and memory performance data for the combined instances of a given process. If events are required, they need to be configured through thresholds on performance data collection templates.

One of the advantages of using performance data collection templates, driven by zencommand, is that you control the data collection interval at the Data Source level. If performance data is collected using SNMP, there is a single polling interval (default 5 mins) for all data collected by the zenperfsnmp daemon.

SNMP is the simple, default method of discovering and monitoring processes and is used by Zenoss's zenmodeler and zenprocess daemons, relying on the Host Resources MIB. The zenprocess daemon has the advantage of very low administrator setup time as performance information is automatically gathered for monitored processes and events are automatically generated if a process is no longer detected. Provided targets support SNMP and Host Resources, there is no agent setup beyond basic configuration of the SNMP agent. The negative aspect of using the built-in Zenoss methods to configure, discover and monitor processes, is that they are still a little “quirky” and do not always deliver the results expected.

For environments where SNMP agent configuration skills exist, the net-SNMP agent can be configured well beyond the ability of the Host Resources MIB by using the UCD-SNMP-MIB process monitoring table. Events can be generated by incorporating the DisMan Event MIB and automatic recovery actions can also be enabled at the agent. For time critical process monitoring, this should be the most responsive solution as monitoring and action can both be taken at the monitored device; there is no polling interval between Zenoss manager and managed device before an event is received. The negative side of extensive agent configuration is that it really only provides event information; there is no performance data provided by this solution.

In practise, some organisation may deploy combinations of all these process monitoring techniques, in order to satisfy their requirements.

References

1. net-SNMP SNMP agent from http://www.net-snmp.org/
2. Host Resources MIB, RFC 2790 obsoletes RFC 1514 - http://www.ietf.org/rfc/rfc2790.txt and http://www.ietf.org/rfc/rfc1514.txt
3. UCD-SNMP-MIB - http://www.net-snmp.org/docs/mibs/UCD-SNMP-MIB.txt
4. DisMan Event MIB, RFC 2981, http://www.ietf.org/rfc/rfc2981.txt
5. Nagios plugin API - http://nagiosplug.sourceforge.net/developer-guidelines.html#PLUGOUTPUT
6. Zenoss FAQ - http://www.zenoss.com/community/docs/faqs/faq-english/
7. Zenoss HowTo for Zenoss plugins - http://www.zenoss.com/community/docs/howtos/zenoss-plugins
8. Zenoss download site - http://www.zenoss.com/download/links/
9. "Zenoss Event Management", by Jane Curry - http://www.zenoss.com/Members/jcurry/zenoss_event_management_paper.pdf/view
10. "Learning Python" by Mark Lutz, published by O'Reilly
11. Zenoss Administration Guide - http://www.zenoss.com/community/docs

Acknowledgements