



Integration between Zenoss and IBM Tivoli Enterprise Console (TEC)

Draft

January 2010

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 the integration of events from Zenoss to IBM's Tivoli Enterprise Console (TEC).

Zenoss is an open source systems and network management offering that first became available in 2006. It provides discovery, availability monitoring, performance management, reporting and events, for networking devices such as routers and switches, and for Unix / Linux / Windows servers.

IBM's Tivoli Enterprise Console is a mature product that dates from the early 1990s; IBM is encouraging customers to replace TEC with their more recent Netcool/OMNIBus but there are a great number of organisations still using TEC and any integration technique into TEC can also be accommodated by Netcool/OMNIBus.

The paper is not intended as a full exposition of either Zenoss or TEC but it does include sufficient discussion of their respective architectures to allow a reasonable understanding of the integration techniques.

The paper focuses on forwarding events from Zenoss to TEC. It would be possible to create a solution to send events from TEC to Zenoss, using SNMP TRAPs, but event flow in this direction seems less likely for most organisations.

This paper was written using Zenoss Core 2.5.1 and TEC 3.9 Fixpack 4.

Notations

Throughout this exercise guide, text to be typed or menu options to be selected will be highlighted by *italics*. Important points to take note of will be shown in **bold**.

Table of Contents

1	Introduction.....	4
2	Zenoss event architecture.....	6
2.1	Event reception.....	6
2.2	Event classes in Zenoss.....	6
2.3	Processing events in Zenoss.....	7
2.4	Automation associated with events.....	9
2.5	Detecting duplicate events.....	10
2.6	Clearing events.....	10
2.7	Events database.....	11
2.8	Zenoss Event Console.....	11
2.9	Generating test events with Zenoss.....	12
3	TEC architecture.....	14
3.1	Event reception.....	14
3.2	TEC classes.....	15
3.3	Processing TEC events.....	17
3.4	Detecting duplicate events with TEC.....	17
3.5	Clearing events.....	18
3.6	TEC rulebases.....	19
3.7	The TEC events database.....	21
3.8	The TEC Event Console.....	21
4	Forwarding events from Zenoss to TEC.....	23
4.1	Elements of the solution.....	23
4.2	Generic TEC configuration.....	23
4.3	Zenoss / TEC configuration using an event command.....	23
4.3.1	TEC configuration.....	24
4.3.2	Zenoss configuration.....	25
4.3.3	Testing the event command solution.....	27
4.3.4	Debugging hints.....	28
4.4	Zenoss / TEC configuration using a page alert.....	29
4.4.1	TEC configuration.....	30
4.4.2	Zenoss configuration.....	35
4.4.3	Testing the page solution.....	38
5	Conclusions.....	43

1 Introduction

There are many different solutions to “monitor” and “manage”, “systems” and “networks”. The quotation marks are deliberate. Different organisation may have very different definitions for each. For the purposes of this paper, “networks” are defined as devices such as:

- routers
- switches
- firewalls
- UPS devices

Anything that can be ping'ed is a potential candidate for some level of management and the availability of a Simple Network Management Protocol (SNMP) agent greatly enhances the manageability of a device.

“Systems” span an even wider domain but would certainly include:

- Unix
- Linux
- Windows

Again, anything ping'able can be managed to a certain extent. The presence of an SNMP agent, Secure Shell (ssh), the Windows Management Instrumentation (WMI) interface for Windows systems, or other open or proprietary agents on the target, will greatly increase its manageability.

The extent of “management” depends on an organisation's requirements. It may include:

- Discovery, perhaps with a network topology diagram
- Inventory, perhaps in a Configuration Management Database (CMDB)
- Availability monitoring – ping, SNMP, port-sniffing, process checking, ...
- Problem management – receiving, processing and reacting to events
- Performance monitoring, thresholding, graphing and reporting
- Configuration management

Zenoss provides a single package that delivers all these features, other than configuration management. IBM offers a more modular solution such as:

- IBM Tivoli Monitoring (ITM) for systems management
- NetView (old product) or IBM Tivoli Network Management (ITNM) (new product) for network management

- Tivoli Enterprise Console (TEC) (old product) or Netcool/OMNIbus (new product) for problem management
- Tivoli Common Reporting (TCR) for reporting and graphing
- Tivoli Configuration Manager (TCM) (old product) or Tivoli Provisioning Manager (TPM) (new product) for configuration management

One major difference between the two suppliers is that the IBM solution relies on deploying proprietary agents for most of their products, whereas Zenoss assumes the presence of “native” agents as part of the target Operating System; for example, an SNMP agent, WMI on a Windows system, syslog on Unix / Linux machines. There are advantages and disadvantages to both approaches. The proprietary agent potentially offers more control but suffers from the management overhead of distributing and maintaining the IBM agents and ensuring that they “play nicely together”.

This paper is only going to examine problem management; strictly, the forwarding of events from Zenoss to TEC. Fundamentally, both solutions offer:

- Native event generation
- Capture and transformation from the native format to TEC / Zenoss format
- Processing of the event which may include changing the incoming event or relating it to other events
- Detecting “duplicate” events
- Automatic action in response to an event
- Storage of the event in a database (relational in both cases)
- Closing events
- An event console

2 Zenoss event architecture

The Zenoss event architecture has elements to receive events, process events and display events.

2.1 Event reception

Zenoss has a number of ways of collecting events. Some are generated internally by Zenoss itself when an availability monitoring daemon detects a problem, as shown in Table 2.1.

Zenoss daemon	Example of when event generated
zenping	ping failure on interface
zendisc	new device discovered
zenstatus	TCP service unavailable
zenprocess	process unavailable
zenwin	Windows service failed
zenperfsnmp	SNMP performance data collection threshold / failure
zencommand	ssh command detected a problem

Table 2.1.: Events generated by Zenoss itself

Zenoss also has three daemons specifically for collecting and interpreting events from external devices. The Zenoss daemons accept native events using well-known ports and then reformats the event to the Zenoss event format.

Zenoss daemon	Example of when event generated
zensyslog	processes syslog events received on UDP/514 (default)
zeneventlog	processes Windows events received using WMI (TCP/135, 139 & 445 and others)
zentrap	processes SNMP TRAPs received on UDP/162

Table 2.2.: External events captured by specialised Zenoss daemons

2.2 Event classes in Zenoss

An event received from an external source is processed through several different mechanisms to assign it to an **event class**, which is the primary event field that determines what transforms will be applied to the event and what automations may take place.

The types of events that Zenoss can process are organised in an object-oriented hierarchy where subclasses of an event class inherit the characteristics of their parent. For example, the Zenoss event class **/Archive** has the **zEventAction**

zProperty set to **history**; this means that all events of this class and any subclasses will automatically be sent to the history table of the events database. Similarly, there is an event class of **Ignore** whose zEventAction is set to **drop**; all events matching this class and its subclasses will be dropped on reception and not stored in the events database.

Zenoss ships with a large number of predefined classes which can be easily modified or augmented. The class definitions are held in Zenoss's Zope Object Database (ZODB) which is the configuration management database (CMDB) used for all Zenoss configuration information (Zenoss is built on the python-based Zope web application server).

2.3 Processing events in Zenoss

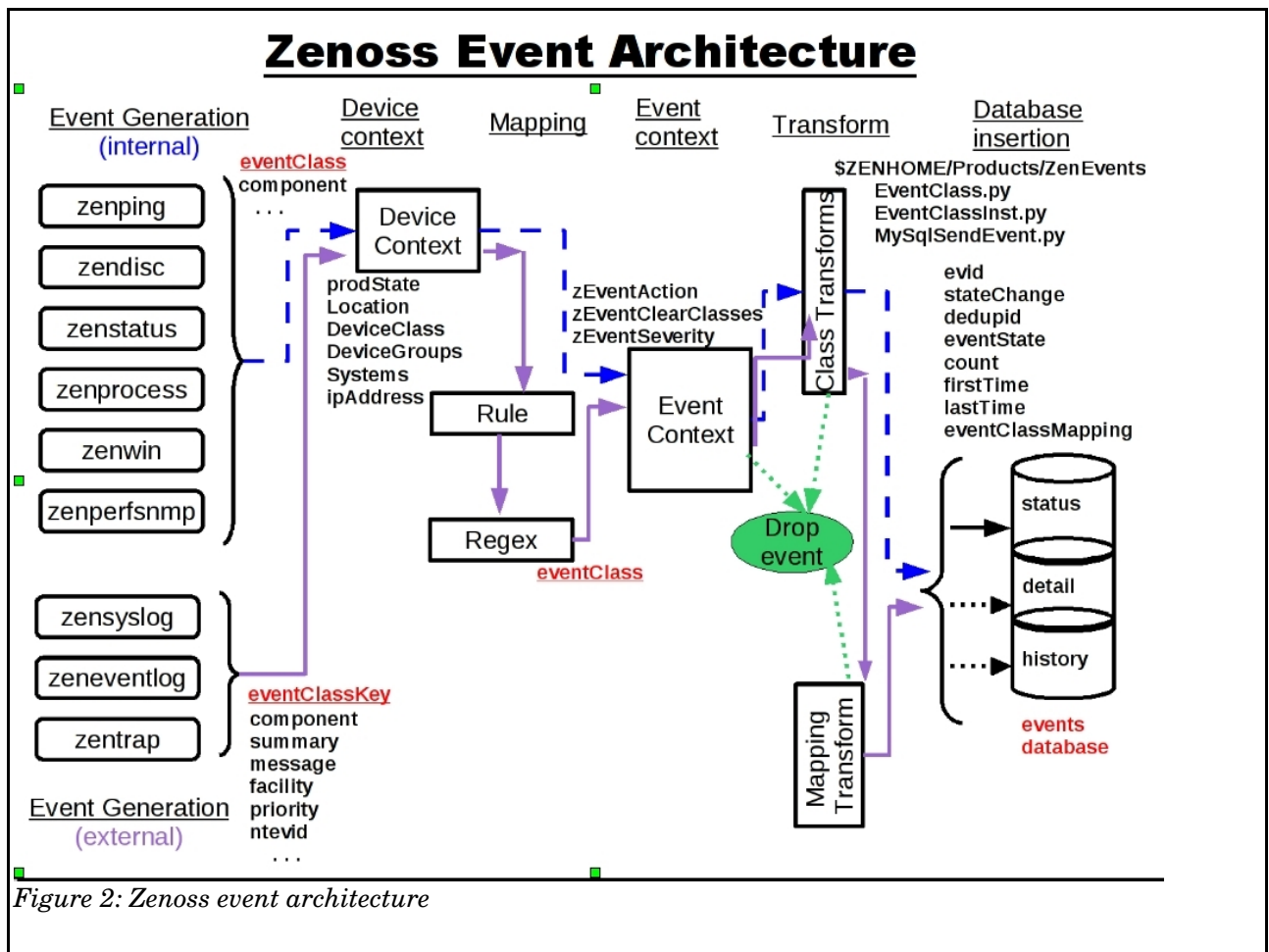
Event processing can range from simple substitution of an event field such as the **summary**, through to complex programs, written in Python. Such **transforms** can be applied to a specific type of external event as a **Mapping Transform**, and to all events of a particular class and its subclasses (**Class Transform**).

The default fields of a Zenoss event are shown in Figure 1 below.

Field	Value
dedupid	zenoss.skills-1st.co.uk sshd 5 PAM audit_log_acct_message() failed: Operation not permitted
evid	0a00008337aba1e7e368ce5
device	zenoss.skills-1st.co.uk
component	sshd
eventClass	/Unknown
eventKey	
summary	PAM audit_log_acct_message() failed: Operation not permitted
message	PAM audit_log_acct_message() failed: Operation not permitted
severity	5
eventState	1
eventClassKey	sshd
eventGroup	syslog
stateChange	2009/01/14 16:05:04.000
firstTime	2009/01/07 11:30:47.000
lastTime	2009/01/14 09:06:01.000
count	2
prodState	1000
suppid	
manager	localhost
agent	zensyslog
DeviceClass	/Server/Linux
Location	/Cedar_Chase
Systems	
DeviceGroups	
ipAddress	10.0.0.131
facility	authpriv
priority	2
nteid	0
ownerid	admin
clearid	
DevicePriority	3
eventClassMapping	
monitor	localhost

Figure 1: Default Zenoss event fields

Figure 2 shows the Zenoss event architecture upto the point where a new event is inserted into the events database,



Since Zenoss builds its own network topology of the devices it discovers, it is able to automatically suppress events from devices behind a single point of failure. It will also suppress higher-level monitoring events from a specific device if lower-level monitoring has failed. For example, if the SNMP agent is down then process monitoring events (which rely on SNMP) will be suppressed. If a device does not respond to ping (that is configured to respond to ping), then all higher-level events will be suppressed.

2.4 Automation associated with events

Automatic actions associated with an incoming event are executed asynchronously by Zenoss's **zenactions** daemon, which runs every minute by default. There are two different types of action:

- Alerts email or paging, executed per user or group
- Event commands any shellscript (which could run perl, python, ...)

Both mechanisms have the same extensive filtering capability to determine what events will actually generate alerts or run event commands, and whether

a corresponding alert / command should be executed when an event clears. Fields from the event can be passed to the alert / command.

The **alert_state** table of the events database ensures that a configured action only runs on the first occurrence of an event and not when duplicate events are received.

2.5 Detecting duplicate events

Zenoss will automatically detect duplicate events, based on the following fields being the same:

- device
- component
- eventClass
- eventKey
- severity
- summary (only if eventKey is blank)

Duplicate events are automatically dropped and the initial event has its **count** field incremented and its **lastTime** field updated.

2.6 Clearing events

Events can be cleared (moved to the history table of the events database) by several different mechanisms:

- The event context of an event class, configured by the **zEventAction** zProperty, can be defaulted to **history** (as described in the above example for events of class /Archive).
- An event transform, executed when the event is first processed, can override the default event context by setting **evt._action = history**
- There is a built-in clearing mechanism in Zenoss whereby any incoming event with a severity of **Clear** will clear any other event in the status table of the events database, which has the following fields the same:
 - eventClass
 - device
 - component
- This automatic clearing mechanism can be extended so that a clear severity event **also** clears events with **other** specified eventClass fields (provided the device and component fields are also the same). Implementation is through the **zEventClearClasses** event context zProperty, or in a transform by setting **evt._clearClasses** .

- Events can be manually cleared by a user through the Event Console
- The Zenoss Event Manager can be configured to automatically clear events of a certain severity after a given length of time. By default, events that are **not** of severity **Critical** or **Error** are automatically cleared after 4 hours.

2.7 Events database

Events are held in a MySQL database, called **events**, the main tables of which are:

- **status** for active events
- **history** for closed events
- **detail** for user-defined event fields
- **alert_state** for detecting whether an alert has already been generated for a duplicate event

2.8 Zenoss Event Console

The AJAX-based Zenoss Graphical User Interface offers an Event Console which provides several different views of the status and history event database tables. Users can view all events, events specific to a device, or all instances of a particular event class. By default, active events are shown; a link at the bottom left will switch to viewing cleared events in the history table.

Status	Severity	Event Class	Summary	First Seen	Last Seen	Count	Component	Device	Owner
⚠	CRITICAL	/Cmd/Fail	Availability test - status CRITICAL	2010-01-25 12:33:12	2010-01-25 17:46:27	295	Availability	rt38.class.examp	
⚠	Warning	/Status/Ping	ip 10.0.0.1 is down	2010-01-25 17:35:12	2010-01-25 17:46:12	12		cisco.skills.1st.co	
⚠	Warning	/Status/Ping	ip 172.16.222.38 is down	2010-01-25 12:33:11	2010-01-25 17:46:12	314		rt38.class.examp	
⚠	Warning	/Status/Ping	ip 10.0.0.94 is down	2010-01-25 17:35:12	2010-01-25 17:46:12	12		10.0.0.94	
⚠	CRITICAL	/Cmd/Fail	Availability test - status CRITICAL	2010-01-25 17:43:37	2010-01-25 17:45:42	3	Availability	win2003.class.ex	
⚠	Warning	/Status/Ping	ip 172.16.222.203 is down	2010-01-11 12:28:48	2010-01-25 17:38:12	3333		win2003.class.ex	
⚠	Warning	/Status/OSProcess	Process not running: sshd	2010-01-20 17:07:16	2010-01-25 17:46:25	724	sshd	lotschy.skills.1st.c	
⚠	Warning	/Status/OSProcess	Process not running: bin.bash..fred	2009-12-22 10:36:34	2010-01-25 17:45:30	4340	bin.bash..fred	zenoss.class.exar	
⚠	Warning	/Status/Snmp	Unable to read processes on device win2003.class.examp	2010-01-05 14:59:52	2010-01-25 13:03:37	295	process	win2003.class.ex	
⚠	Warning	/Status/Snmp	SNMP agent down	2010-01-21 13:35:14	2010-01-25 12:33:17	234	snmp	rt38.class.examp	
⚠	Warning	/Status/OSProcess	Process not running: calc.exe	2010-01-04 11:18:21	2010-01-05 10:42:59	19	calc.exe	win2003.class.ex	
⚠	Warning	/Perf/Interface	threshold of high utilization exceeded: current value 11520	2010-01-25 17:38:08	2010-01-25 17:43:05	2	Z	group-100-s1.clas	
⚠	Warning	/Perf/Interface	threshold of high utilization exceeded: current value 12531	2010-01-25 12:43:02	2010-01-25 17:43:04	61	21	group-100-s1.clas	
⚠	Warning	/Perf/Interface	threshold of high utilization exceeded: current value 11526	2010-01-25 17:38:03	2010-01-25 17:43:04	2	FastEthernet1/0	group-100-r3.clas	
⚠	Warning	/Perf/Interface	threshold of high utilization exceeded: current value 11526	2010-01-25 17:38:03	2010-01-25 17:43:04	2	FastEthernet1/2	group-100-r3.clas	
⚠	Warning	/Perf/Interface	threshold of high utilization exceeded: current value 12470	2010-01-25 12:43:02	2010-01-25 17:38:08	60	19	group-100-s1.clas	
⚠	Warning	/Perf/Interface	threshold of high utilization exceeded: current value 12436	2010-01-25 12:53:03	2010-01-25 17:38:08	16	B	group-100-s1.clas	

Figure 3: Zenoss Event Console

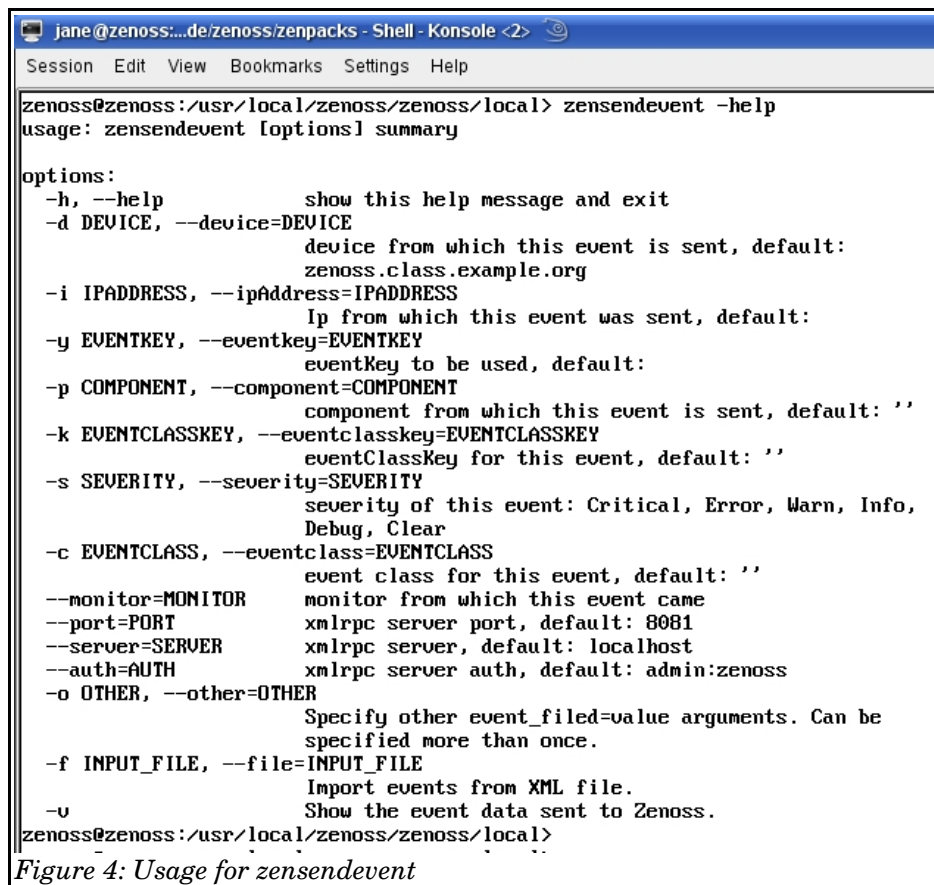
Double-clicking on an event results in a separate window displaying all fields of the event.

2.9 Generating test events with Zenoss

When configuring events within Zenoss it is necessary to be able to generate test events to exercise any new configuration. The Event Console GUI provides a simple dialogue using the + icon seen at the top of Figure 3. You are prompted for the following fields for an event:

- Summary
- Device
- Component
- Severity
- Event Class Key
- Event Class

For repetitive testing, the command line interface, **zensendevent** is more appropriate:



```
jane@zenoss:...de/zenoss/zenpacks - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
zenoss@zenoss:/usr/local/zenoss/zenoss/local> zensendevent -help
usage: zensendevent [options] summary

options:
  -h, --help                show this help message and exit
  -d DEVICE, --device=DEVICE
                           device from which this event is sent, default:
                           zenoss.class.example.org
  -i IPADDRESS, --ipAddress=IPADDRESS
                           Ip from which this event was sent, default:
  -y EVENTKEY, --eventkey=EVENTKEY
                           eventKey to be used, default:
  -p COMPONENT, --component=COMPONENT
                           component from which this event is sent, default: ''
  -k EVENTCLASSKEY, --eventclasskey=EVENTCLASSKEY
                           eventClassKey for this event, default: ''
  -s SEVERITY, --severity=SEVERITY
                           severity of this event: Critical, Error, Warn, Info,
                           Debug, Clear
  -c EVENTCLASS, --eventclass=EVENTCLASS
                           event class for this event, default: ''
  --monitor=MONITOR         monitor from which this event came
  --port=PORT               xmlrpc server port, default: 8081
  --server=SERVER           xmlrpc server, default: localhost
  --auth=AUTH               xmlrpc server auth, default: admin:zenoss
  -o OTHER, --other=OTHER
                           Specify other event_filed=value arguments. Can be
                           specified more than once.
  -f INPUT_FILE, --file=INPUT_FILE
                           Import events from XML file.
  -v                        Show the event data sent to Zenoss.
zenoss@zenoss:/usr/local/zenoss/zenoss/local>
```

Figure 4: Usage for zensendevent

Note that any command line Zenoss work should always be performed as the **zenoss** user. This user is created when Zenoss is installed but logins are disabled so the usual procedure is to su to root and then su to zenoss:

```
su                                and provide the root password
su - zenoss
zensendevent -d zenoss.class.example.org -s Critical -k badnews -p TestComp This is bad
news 1
```

For much greater detail on Zenoss event architecture, get the paper “Zenoss Event Management” from [http://www.skills-1st.co.uk/papers/jane/zenoss event management paper.pdf](http://www.skills-1st.co.uk/papers/jane/zenoss_event_management_paper.pdf).

3 TEC architecture

There are many similarities between the Zenoss event architecture and TEC. Both have event reception, event processing, automatic actions and an event console, and both keep their events in a relational database. Both use an object-oriented hierarchy of event classes to define event types.

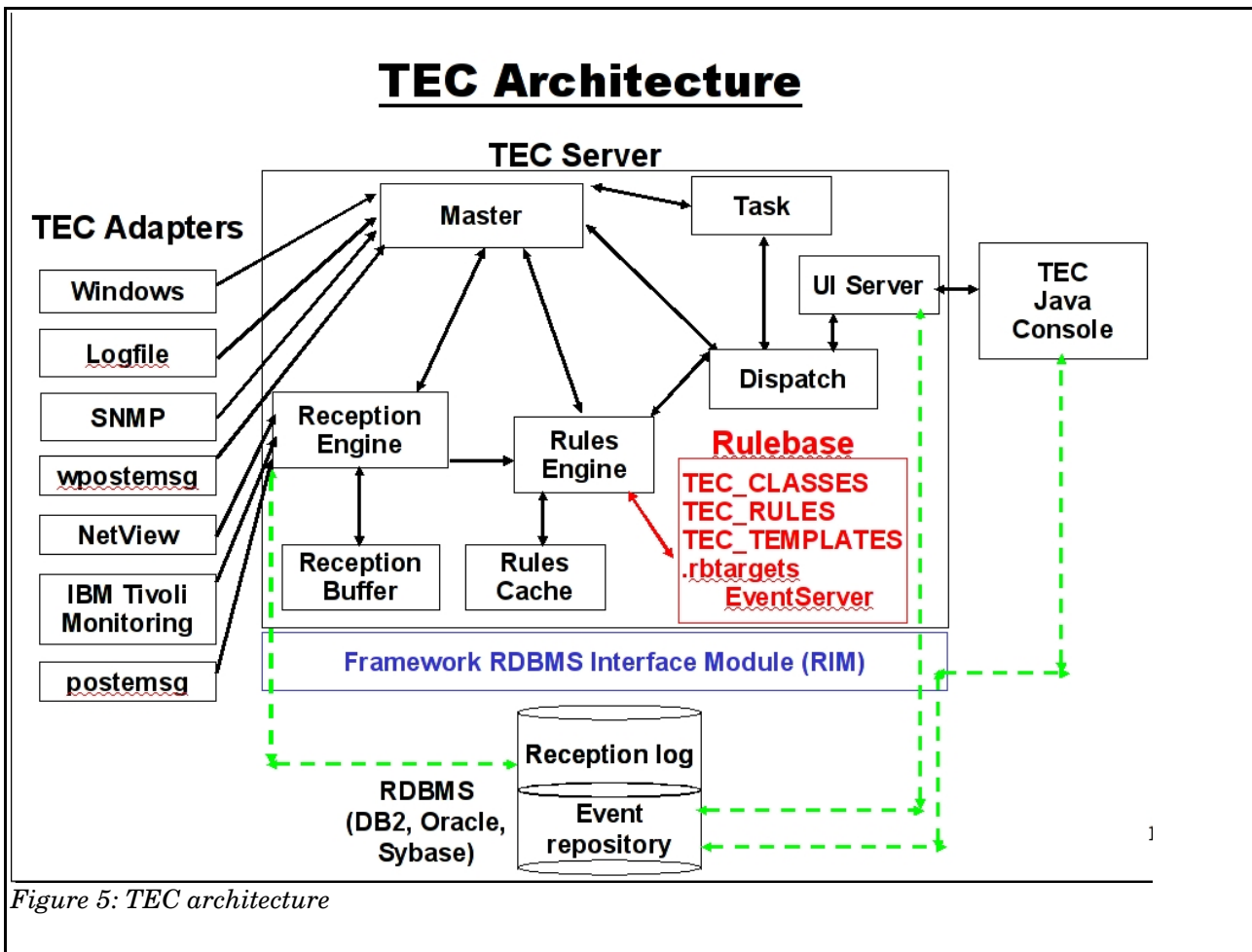


Figure 5: TEC architecture

3.1 Event reception

Event reception is handled by **TEC Adapters**. These are proprietary agents that are installed on target devices, typically one per application type. A TEC adapter must:

- Detect a native event
- Format that event into a TEC format event
- Forward the event to the TEC Server (also known as the Event Server)

IBM provides a large number of TEC adapters to interface with most of their products, plus some generic adapters to collect events from Windows, Unix

syslogs (and indeed any textual file) and SNMP. Although these TEC adapters need to be installed and maintained the IBM Tivoli Framework provides a very simple, scalable method for deployment.

Some adapters forward events over the network using Tivoli Framework communication protocols (**TME** or **secure** adapters); other adapters are built as **non-TME** or **non-secure** – they simply use a TCP socket pairing of <IP address> plus <port>. Strictly, TME events are received at the Event Server by the **master, tec_server** process which forwards them to the reception engine; non-TME events are received directly by the reception engine. In order to use TME adapters, a device must have the Tivoli Framework endpoint code installed.

IBM provides two “one-line TEC adapters” - a command line to generate either a TME or non-TME event. These are **wpostemsg** (or wpostzmsg) for TME events and **postemsg** (or postzmsg) for non-TME events. The format for each is very similar, the difference being that the non-TME variant requires a **-f <configuration file>** parameter which specifies at least the IP address and receiving port of the Event Server. This configuration file can also contain any other legal parameter for a TEC adapter configuration file – see the TEC Adapters Guide for a full list. The syntax for postemsg is:

```
postemsg -f config_file [ -m message] [ -r severity] [attribute =value...]  
class source
```

Any event can be constructed using multiple **attribute=value** elements. The postemsg command must have the event **class** as the next-to-last parameter and the event **source** as the final parameter. The source field specifies the type of adapter that the event emanated from; examples would include LOGFILE, NV6K and SNMP.

postemsg is a standalone binary that is shipped with TEC for various different architectures (such as Windows, Linux, AIX, ...). It has no other requirements other than TCP/IP communications.

The (w)postemsg commands are often used to generate test events when verifying Event Server configurations.

3.2 TEC classes

All TEC events inherit from the **base event** (called **EVENT**) which is defined in **root.baroc**. The base event is shown below:

```

TEC_CLASS :
    EVENT
    DEFINES {
        server_handle:  INTEGER, parse = no;
        date_reception: INT32, parse = no;
        event_handle:   INTEGER, parse = no;
        source:         STRING;
        sub_source:     STRING;
        origin:         STRING;
        sub_origin:     STRING;
        hostname:       STRING;
        adapter_host:   STRING;
        date:           STRING;
        status:         STATUS, default=OPEN;
        administrator:  STRING, parse = no;
        acl:            LIST_OF STRING,
                       default = [admin],
                       parse = no;

        credibility:   INTEGER, default = 1, parse = no;
        severity:      SEVERITY, default = WARNING;
        msg:           STRING;
        msg_catalog:   STRING;
        msg_index:     INTEGER;
        duration:      INTEGER, parse = no;
        num_actions:   INTEGER, parse = no;
        repeat_count:  INTEGER;
        cause_date_reception: INT32, parse = no;
        cause_event_handle:  INTEGER, parse = no;
        server_path:   LIST_OF STRING;
    };

END

```

Figure 6: The TEC Base Event, called EVENT

As with Zenoss, a hierarchy of event classes is constructed where event subclasses inherit the characteristics of their parent class. Classes must be defined before they can be used; they are written in the TEC BAROC (BASic Recorder of Objects In C) language and must be in a file with a **.baroc** suffix. Typically, there is a baroc file for each TEC adapter type, plus some local class configuration files.

3.3 Processing TEC events

The central TEC Server is generally implemented on a dedicated machine and consists of a number of separate processes. Typically the TEC relational database will also be installed on the same system.

It is the function of the TEC reception engine to receive events and ensure that they match a defined TEC class. The **wtdumpprl** command can be used to display all events that arrive at the reception engine, even if they are discarded at this stage. This is a very useful debugging command.

Once an event is accepted, it moves to the rules engine. The rules engine is configured by one or more rules files (that must have a **.rls** extension). Typically each TEC adapter will come with a rules file; an organisation may well also write their own rules files. These files may contain rules either in the TEC rules meta-language or in Prolog (the native language of TEC). An event under analysis will be compared against **all** rules in the rulebase and any event transformations or actions will be executed. Simple primitive actions (such as modifying the msg attribute) will be executed by the TEC dispatch engine; long-running actions such as scripts or Tivoli Tasks will be run asynchronously by the TEC task engine.

Thus there is a fairly close analogy between Zenoss transforms and TEC rules processed by the dispatch engine, which make “simple” modifications to an event, and between Zenoss's zenactions daemon and the TEC rules processed by the task engine which implement script-based automation.

3.4 Detecting duplicate events with TEC

TEC uses a combination of information held in the class **.baroc** files and TEC rules to detect duplicate events. This makes TEC more flexible but more effort to configure. A duplicate event is defined as an event with the same event **class**, plus all event attributes with the **dup_detect** facet set in a **.baroc** file, must also be the same. For example, here is an entry in a file called *zenoss.baroc*:

```
TEC_CLASS :
    Zenoss_Base ISA EVENT
    DEFINES {
        source: default= "ZENOSS";
        sub_source: dup_detect=yes;
        sub_origin: dup_detect=yes;
        adapter_host: default= "N/A";
        msg_catalog: default= "none";
        msg_index: default= 0;
        repeat_count: default= 0;
        severity: default = WARNING, dup_detect=yes;
        hostname: dup_detect=yes;
    };
```

END

The class **Zenoss_Base** inherits all the characteristics of the base event, *EVENT*. The *sub_source*, *sub_origin*, *hostname* and *severity* attributes (that all exist in the base event) have an overriding definition here that sets the **dup_detect facet**.

These class definitions alone, do nothing. There must also be a rules file that looks for duplicate events and acts on them. Typically, such a rule would add to the *repeat_count* attribute of the original event and drop the duplicate event.

```
rule: filter_duplicate_zenoss: (  
  description: 'Filter duplicates for Zenoss events',  
  event:_event of_class 'Zenoss_Base',  
  action: filter: (  
    first_duplicate(_event, event: _dup_ev  
      where [  
        status: outside ['CLOSED']  
      ],  
      _event - 600 - 600),  
    add_to_repeat_count(_dup_ev, 1),  
    drop_received_event  
  )  
).
```

This rule checks the event under analysis for a class of *Zenoss_Base*. If the event is of this class, the events database is searched for a duplicate event, that does not have a *status* of *CLOSED*, within the last 600 seconds. If an event is found in the database, its *repeat_count* attribute is incremented and the event under analysis is dropped.

3.5 Clearing events

Closed or cleared events have a totally different architecture in TEC and Zenoss.

TEC has a **status** attribute in the base event; it can take values of **OPEN**, **ACK**, **RESPONSE** and **CLOSED**. Zenoss events do have a status field but this takes the values of **New** (0), **Acknowledged** (1) and **Suppressed** (2) – there is no **Closed** (Suppressed is typically used to hide events from devices behind a single-point-of-failure).

Zenoss has a separate **history** table in the RDBMS database for closed events. TEC has a single table, the event repository, for all events that have been received and processed; closed events are simply denoted by their status attribute.

TEC has no automatic, built-in clearing rules as Zenoss does; however there are a number of different methods for closing events.

- It is possible, though unusual, for a TEC adapter to send an event to the Event Server, with a status of CLOSED.
- A simple TEC rule could set the status attribute of any specified event to CLOSED.
- TEC has a rule primitive that allows the TEC database to be searched for “associated” events. The primitive will then create a link between a **causal** event and an **effect** event. A subsequent rule may then be activated by a “good news” event which closes the earlier causal event **and** searches the database for associated effect events, and closes those too. This is rather analogous to the Zenoss **zEventClearClasses** event context zProperty but the TEC mechanism is potentially more flexible (although it requires more effort to code).
- Events can be manually cleared by a user through the TEC Event Console
- The TEC Server product includes a number of ruleset files including **cleanup.rls** which, by default, automatically closes events of severity HARMLESS or UNKNOWN after 48 hours.

3.6 TEC rulebases

All acceptable events and the rules to process those events, are defined on the Event Server in a directory hierarchy of files, known as a **rulebase**. An Event Server may have several different rulebases defined but only one will be active at any given time. A rulebase has the following directory structure:

- TEC_CLASSES files that define event classes and subclasses. Subscript is **.baroc**.
- TEC_RULES files containing event processing rules. Subscript is **.rls**.
- TEC-TEMPLATES compiled Prolog files containing primitives
- **.rbtargets/EventServer** **Note** the dot at the beginning! This directory contains a copy of the TEC_CLASSES, TEC_RULES and TEC_TEMPLATES directories and is what the Event Server is actually loaded from (there used to be the possibility of other types of Event Server but this is now deprecated).

A strict process must be adhered to when manipulating a rulebase. Source files for a rulebase should be held in a working directory outside the base rulebase directory. Baroc files and rulebase files are then **imported** in to a

rulebase and **compiled**. Once compilation is successful, the rules are imported on to the EventServer rulebase **target**. The rulebase is then **loaded** and, if the baroc files have changed at all, or a new rules file has been added, then the Event Server must be **stopped** and **restarted**.

As an example, assume a rulebase called *myrulebase* already exists with a home directory of */usr/local/Tivoli/TEC_rb/myrulebase*. A new baroc file, *zenoss.baroc*, and a rules file, *zenoss.rls*, are to be added to the existing rulebase. These source files are in */usr/local/Tivoli/TEC_rb/sources*.

```
cd /usr/local/Tivoli/TEC_rb/sources
wlsrb -d                shows existing rulebases & directories
wlscurrb              shows the rulebase currently loaded
wrb -imprbclass zenoss.baroc myrulebase  imports baroc file into
myrulebase
wrb -imprbrule zenoss.rls myrulebase     imports rules file into
myrulebase
wrb -imptgtrule zenoss EventServer myrulebase imports a rules file to
the                                     EventServer
target. Note no .rls                    suffix
here.
wrb -comprules myrulebase                compile the rulebase
wrb -loadrb myrulebase                   load the rulebase
wstopesvr                               stop the Event Server
wstartesvr                               start the Event Server
```

Sometimes it is helpful to run extra *wrb -comprules* commands to check that no errors have yet been introduced. Note that baroc files must be imported before any rules files which make use of classes defined in those baroc files.

If something goes wrong, again a strict procedure should be followed to remove offending files and reimport them. If class baroc files need to be edited, then any dependent rules files must also be removed first. Thus, the procedure would be:

```
wrb -deltgtrule zenoss EventServer myrulebase  Remove the rules file
from the                                     EventServer
target. Note no                               suffix.
.rls suffix.
wrb -delrbrule zenoss myrulebase              Remove the rules file from the
rulebase. Note                                no .rls suffix.
wrb -delrbclass zenoss.baroc myrulebase       Remove the baroc file from the
rulebase. Note presence
of .baroc suffix.
```

3.7 The TEC events database

TEC uses a relational database to store events; the RDBMS can be DB/2, Oracle or Sybase. Strictly, TEC accesses the underlying database through the RDBMS Interface Module (RIM) of the Tivoli Framework (which is a prerequisite before installing a TEC Server).

TEC uses the database to store many configuration aspects of TEC, in addition to the actual events. The database is usually called **tec** and the main tables for holding events are:

- Reception log records all events reaching the reception engine
- Event repository holds all processed events

3.8 The TEC Event Console

The TEC Event Console is a Java application that can present event information at three levels of detail:

- Summary of event groups
- Detailed events, one line per event, for a particular event group
- Detailed information including all event attributes, for a specific event

Event groups is the technique for assigning an event to one or more categories for the purpose of the TEC Console graphical user interface; event groups have no effect on event processing; simply their display to users.

From the Summary, click on the bar representing events in an event group, to get to the details of that group. Note that the detailed window is divided into two halves. The top half (the Working Queue) is where operators should focus. Events selected here can then be manipulated using the buttons in the middle of the display – **Close**, **Acknowledge** and **Details** (to get to the fine detail of a selected event), are the common choices.

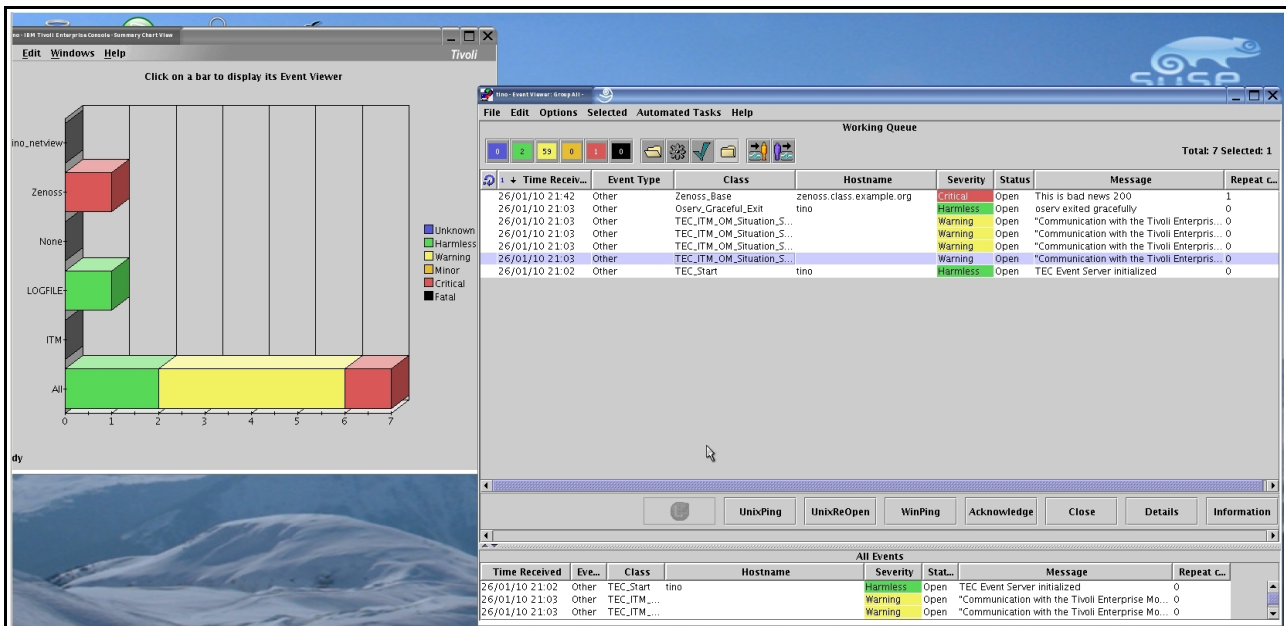


Figure 7: TEC Console showing summary of Event Groups and details for a specific Event Group

The TEC Console is populated by retrieving data from the event repository table in the database. The UI Server process, which is normally installed with the Event Server, provides authorized access between the TEC Console and the events.

4 Forwarding events from Zenoss to TEC

4.1 Elements of the solution

Zenoss has two possible mechanisms for forwarding events to TEC:

- Event command, run by zenactions
- Alert, run by zenactions

Either way, TEC's standalone postmsg utility will drive the communication.

On TEC, a baroc file will be required to interpret event classes from Zenoss and a rules file would be useful to help detect duplicate events and to correlate “good news” events with “bad news” events.

4.2 Generic TEC configuration

Zenoss will use TEC's postmsg command to forward an event to TEC. This requires a TEC **class** parameter and a TEC **source** parameter. The source specifies the type of TEC Adapter that generated the event. By convention, TEC sources use uppercase. To use TEC command-line commands, a user will need to be configured as a **Tivoli Administrator** with the **Senior** role for TEC related areas.

To create the ZENOSS source, with a label of Zenoss, use:

```
wcrtsrc -l Zenoss ZENOSS
```

To facilitate categorizing events for the TEC Console GUI, **Event Groups** can be created and assigned to a user's Console configuration. To create a new event group called *Zenoss*, add a filter to it called *ZENFILTER* to include all events where *source=ZENOSS*, and then list all event groups:

```
wconsole -crteg -n Zenoss -D "Description of Zenoss event group"  
wconsole -addegflt -E Zenoss -n ZENFILTER -D "Desc" -S "source =  
ZENOSS"  
wconsole -lseg
```

Assume that a TEC Console definition already exists for Tivoli user *Jane*. To assign the new event group *Zenoss* to this user, with senior, admin and user roles:

```
wconsole -assigneg -C Jane -E Zenoss -r senior:admin:user
```

Any TEC Console that is open during this configuration, must be restarted before the changes are seen.

4.3 Zenoss / TEC configuration using an event command

The example developed in this section uses a simple command called directly from the event command interface. It demonstrates substituting some fields from the Zenoss event into the postmsg command, using TALEX expressions (Template Attribute Language Expression Syntax). The alerting solution

presented in the next section uses a more complex intermediate script to perform extra processing; it would be perfectly possible to do something similar for an event command.

4.3.1 TEC configuration

This example incurs minimal configuration on the TEC Server. A single new event class, *Zenoss_Base*, will be created which has no new attributes. Existing attributes of the base event will be used to pass relevant Zenoss event fields.

Zenoss Event Field	TEC Event Attribute
evid	sub_origin
device	hostname
ipAddress	origin
summary	msg
component	sub_source

Table 4.1: Mapping of Zenoss event fields to TEC class attributes

In addition to these field / attribute mappings, the **severity** TEC attribute will be coded as the literal "WARNING" and the **adapter_host** attribute will be the literal "zenoss.class.example.org" (the Zenoss server). The **source** in the postmsg command will be the new ZENOSS source and the TEC **class** will be *Zenoss_Base*, which needs defining in a TEC baroc file on the TEC Server.

```

TEC_CLASS :
    Zenoss_Base ISA EVENT
    DEFINES {
        source: default= "ZENOSS";
        sub_source: dup_detect=yes;
        sub_origin: dup_detect=yes;
        adapter_host: default= "N/A";
        msg_catalog: default= "none";
        msg_index: default= 0;
        repeat_count: default= 0;
        severity: default = WARNING, dup_detect=yes;
        hostname: dup_detect=yes;
    };
END

```

Figure 8: zenoss.baroc containing the Zenoss_Base class

Note in Figure 8 that the sub_source, sub_origin, hostname and severity attributes have had the dup_detect facet set for Zenoss_Base, in addition to some default values being set for other attributes. This potentially allows TEC

to detect duplicate events based on events whose class is `Zenoss_Base` and whose original `evid`, `component`, `device` and `severity` fields are the same.

4.3.2 Zenoss configuration

Event commands are configured in the Zenoss GUI from the left-hand *Event Manager* menu, by opening the *Commands* tab. To create a new command, type a unique name (*toTec*) in the box at the bottom of the screen and click *Add*. Once the command shows in the list, click on its name to edit it.

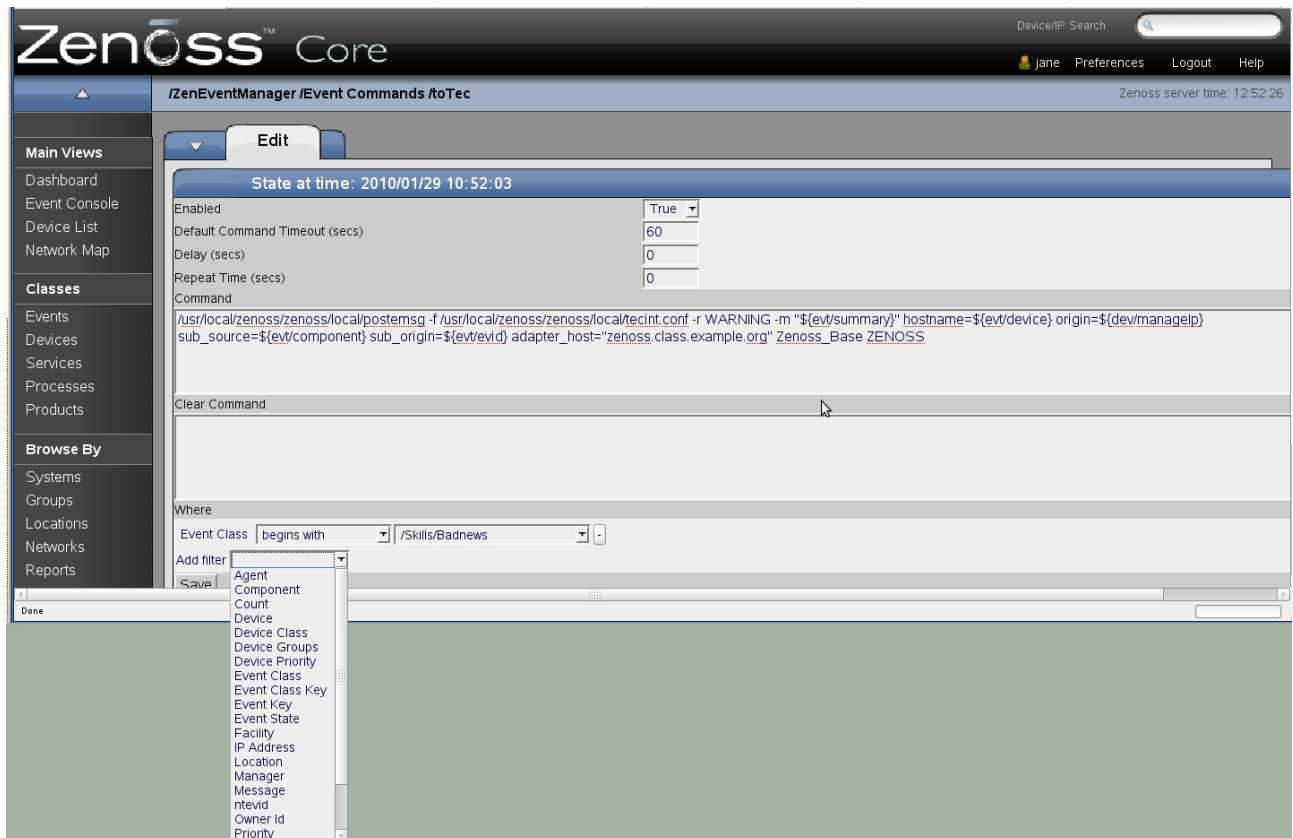


Figure 9: Editing the *toTec* Event Command

Do take note of the *Enabled* field at the top of the configuration screen! The dialogue allows you to type any shell command. There is also an area for running a command when an event is cleared to the history table of the Zenoss events database.

The bottom area of the dialogue allows for filters which can very closely define what events will trigger a command. Multiple filters are logically AND'ed. If a logical OR is required then a separate event command should be created with the same command and the alternative filter(s). Any standard field of a Zenoss event can be employed in a filter. The filter used here checks for the **eventClass** field beginning with */Skills/Badnews*.

The `postmsg` binaries should be available on the TEC EIF disk or image, under *EIFSDK/bin/<architecture>*. Copy the appropriate architecture binary

to the Zenoss system. The **\$ZENHOME** environment variable is setup as part of the zenoss user's environment; for SuSE, this is `/usr/local/zenoss/zenoss`. A convenient practice is to create a subdirectory, *local*, under **\$ZENHOME** for locally-created Zenoss utilities.

The configuration file required by `postemsg` only mandates entries for the resolvable name or IP address of the TEC Server (*ServerLocation*) and the port that the reception engine listens on (*ServerPort*). A TEC Server implemented on a Unix platform usually uses `portmapper` to allocate the port so *ServerPort* takes a zero value; the default port for a Windows-based TEC is 5529.

```
ServerLocation=tino.skills-1st.co.uk
ServerPort=0
BufferEvents=YES
BufEvtPath=/usr/local/zenoss/zenoss/local/tecint.cache
```

Figure 10: /usr/local/zenoss/zenoss/local/tecint.conf configuration file for postemsg

In addition to the mandatory parameters, the example above ensures that events are cached if the TEC Server cannot be contacted, in `/usr/local/zenoss/zenoss/local/tecint.cache`.

Remember that the syntax for the TEC `postemsg` command is:

```
postemsg -f config_file [ -m message] [ -r severity] [attribute =value...] class
source
```

In the event command shown in Figure 9, a fixed class of *Zenoss_Base* is used and the final source parameter is *ZENOSS*.

Event fields from the Zenoss event are substituted into the `postemsg` command using space-separated `<tec_attribute_name> = <value>` pairs.

Many attributes of both the event and the device that caused the event are available for substitution using TALES expressions. These are documented in the Zenoss Administration Guide, Appendix E. The syntax for substitution is:

```
 ${evt/<field>}           for example ${evt/eventClass}
or
 ${dev/<attribute>}       for example ${dev/snmpContact}
```

Literal strings can also be used as attribute values – ensure that any strings with spaces are enclosed in double quotes.

The complete `postemsg` command to be used in the event command (all on one line) will be:

```
/usr/local/zenoss/zenoss/local/postemsg -f
/usr/local/zenoss/zenoss/local/tecint.conf -r WARNING -m "$
{evt/summary}" hostname=${evt/device} origin=${dev/manageIp}
sub_source=${evt/component} sub_origin=${evt/evid}
adapter_host="zenoss.class.example.org" Zenoss_Base ZENOSS
```

Once an event command has been saved, nothing else is required to activate it. zenactions runs every 60 seconds by default and will compare all *New* status events in the status table of the events database, against all enabled write commands. This means that when new event commands are created they will be activated against all existing, old, open events (which can be a bit of a surprise!). It can also be a large load on the Zenoss system. Judicious use of filters in the event command should prevent debilitating action storms.

zenactions logs in the **alert_state** table of the events database when an action has been run for an event. It uses this table to ensure that actions are not run for duplicate events and also to help the clearing logic when a “good news” event clears a previous “bad news” event.

4.3.3 Testing the event command solution

To drive the event command, the following zensendevent command was used on the Zenoss system, as the zenoss user:

```
zensendevent -d zenoss.class.example.org -s Critical -k badnews -p TestComp This is bad news 301
```

The event appears in the Zenoss Event Console as shown in Figure 11.

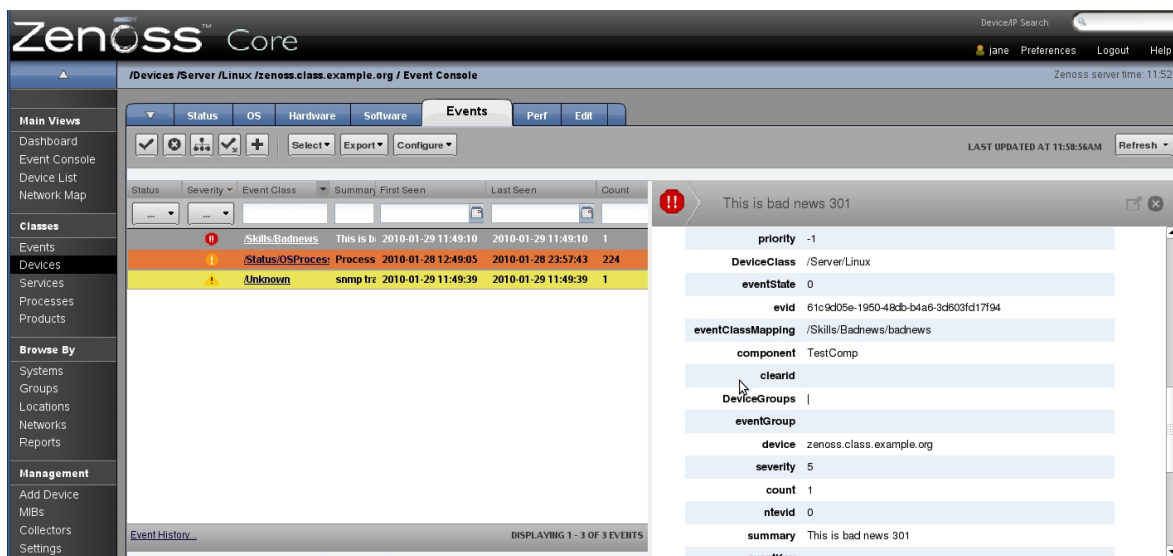


Figure 11: Bad news event in the Zenoss Event Console with detailed event displayed

The Zenoss event is forwarded to TEC by the toTECZenoss event command.

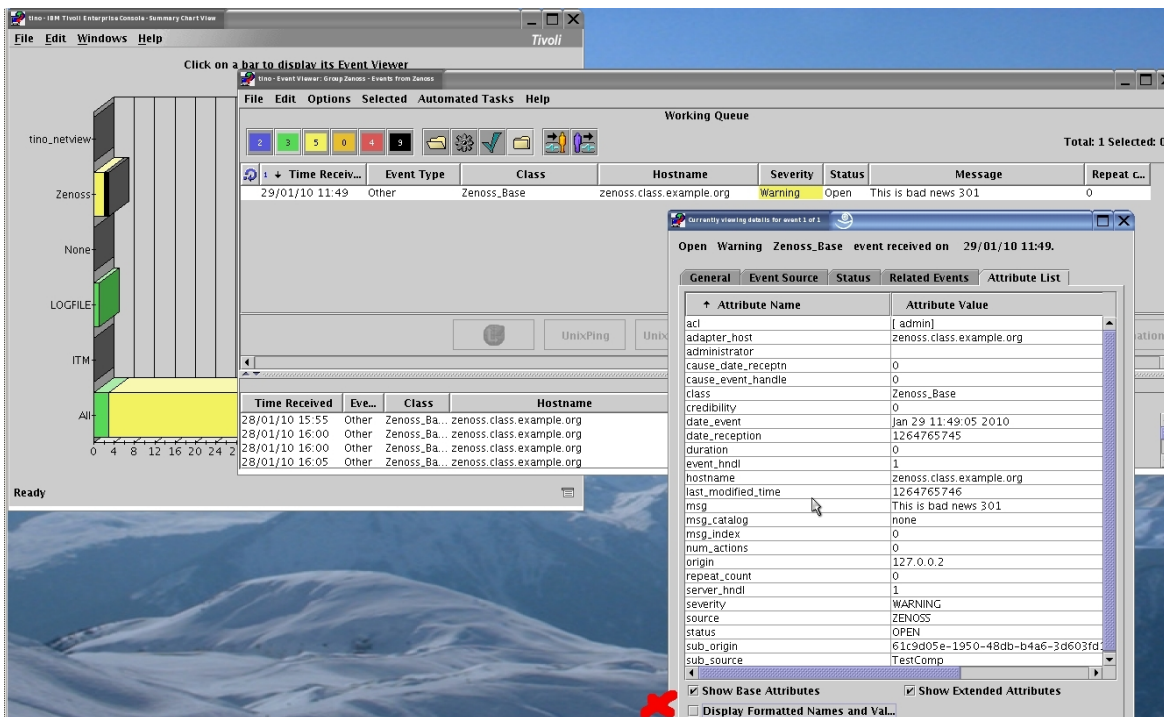


Figure 12: Bad news event in the Zenoss event group in the TEC Console

Each of the Zenoss event fields has been passed to the relevant TEC attributes. Note that the default setting in the TEC detailed event window is to tick the *Display Formatted Names and Values* box. This results in “user-friendly” event attribute names which sometimes do **not** match the actual attribute names in a TEC baroc file. It is recommended that this box be unchecked so that the baroc-defined names are displayed.

4.3.4 Debugging hints

The Zenoss event command is run by the **zenactions** daemon. An excellent debugging aid is to monitor `$ZENHOME/log/zenactions.log`. The default, *Info*, level of debugging is usually adequate but it can be turned up to *Debug* level, if required – use the left-hand *Settings* menu and the *Daemons* tab to view and edit daemon options and to restart them.

```

2010-01-29 11:49:36,789 INFO zen.ZenActions: Running /usr/local/zenoss/zenoss/local/postmsg -f /usr/local/zenoss/zenoss/local/tecint.conf -r
WARNING -m "This is bad news 301" hostname=zenoss.class.example.org origin=127.0.0.2 sub_source=TestComp sub_origin=61c9d05e-1950-48db-b4a6-3d
603fd17f94 adapter_host="zenoss.class.example.org" Zenoss_Base ZENOSS
2010-01-29 11:49:37,091 INFO zen.ZenActions: Running echo "DOWN 2010/01/29 11:49:10.000 2010/01/29 11:49:10.000 1 zenoss.class.example.org 127
.0.0.2 This is bad news 301 " >> /tmp/cndoutput
2010-01-29 11:49:37,489 INFO zen.ZenActions: Processed 4 commands in 4.749600
2010-01-29 11:49:37,525 INFO zen.ZenActions: processed 0 rules in 4.79 secs
2010-01-29 11:50:37,580 INFO zen.ZenActions: Processed 4 commands in 0.025907
2010-01-29 11:50:37,587 INFO zen.ZenActions: processed 0 rules in 0.04 secs
2010-01-29 11:51:37,872 INFO zen.ZenActions: Processed 4 commands in 0.020371
2010-01-29 11:51:37,881 INFO zen.ZenActions: processed 0 rules in 0.03 secs
2010-01-29 11:52:38,174 INFO zen.ZenActions: Processed 4 commands in 0.168341
2010-01-29 11:52:38,175 INFO zen.ZenActions: processed 0 rules in 0.17 secs

```

Figure 13: zenactions.log showing 2 actions for an event – the postmsg command & an echo command

Sometimes zenactions seems to “hiccup” when running actions (this is not particular to postmsg actions). The result is that an action is sometimes run twice in consecutive zenaction processing intervals. This can mean that a duplicate event is sent to TEC. Normally this would not be possible – if zensendevent is used to generate a duplicate event (by Zenoss definitions), it simply adds to the repeat count of the existing duplicate Zenoss event. Event commands are not executed for event duplicates so no duplicate should normally be sent to TEC.

The following figure demonstrates the entry in zenactions.log when a “hiccup” has occurred when processing the tecSend action (described in the next section). An error is generated at 16:17:55 but the tecSend runs successfully at 16:18:55. In practise, the event command is actually run in both intervals, resulting in the duplicate event at TEC.

```

jane@zenoss:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
2010-01-28 16:16:54,634 INFO zen.ZenActions: processed 1 rules in 0.04 secs
2010-01-28 16:17:54,782 INFO zen.ZenActions: Running /usr/local/zenoss/zenoss/local/gen_alert_trap.sh zenoss.class.e
xample.org TestComp "This is bad news Z20"
2010-01-28 16:17:55,008 INFO zen.ZenActions: Running echo "DOWN 2010/01/28 16:17:49.000 2010/01/28 16:17:49.000 1 ze
noss.class.example.org 127.0.0.2 This is bad news Z20 " >> /tmp/cmdoutput
2010-01-28 16:17:55,054 INFO zen.ZenActions: Processed 3 commands in 0.338521
2010-01-28 16:17:55,169 WARNING zen.ZenActions: SELECT firstTime,severity,evid,component,summary,device,message,ipAd
dress,severity,summary,ownerid,stateChange, evid FROM status WHERE (prodState = 1000) and (eventState = 0) and (even
tClass like '/Skills/Badnews:/' ) AND evid NOT IN (SELECT evid FROM alert_state WHERE userid='tec' AND rule='tecSen
d' )
2010-01-28 16:17:55,172 ERROR zen.ZenActions: action:tecSend
Traceback (most recent call last):
  File "/usr/local/zenoss/zenoss/Products/ZenEvents/zenactions.py", line 244, in processRules
    self.processEvent(zem, ar, actfunc)
  File "/usr/local/zenoss/zenoss/Products/ZenEvents/zenactions.py", line 293, in processEvent
    if action(context, data, False):
  File "/usr/local/zenoss/zenoss/Products/ZenEvents/zenactions.py", line 662, in sendPage
    self.dmd.pageCommand()
  File "/usr/local/zenoss/zenoss/Products/ZenUtils/Utils.py", line 693, in sendPage
    response = p.stdout.read()
IOError: [Errno 4] Interrupted system call
2010-01-28 16:17:55,175 INFO zen.ZenActions: processed 1 rules in 0.47 secs
2010-01-28 16:18:55,204 INFO zen.ZenActions: Processed 3 commands in 0.019657
2010-01-28 16:18:55,335 INFO zen.ZenActions: sent page to 12345: severity=4 sub_source=121f65e6-2b2a-4866-b1a7-b0885
0e1cc3b msg="This is bad news Z20" hostname=zenoss.class.example.org origin= sub_origin=TestComp
2010-01-28 16:18:55,364 INFO zen.ZenActions: processed 1 rules in 0.19 secs
83040,1 98%
Shell

```

Figure 14: zenactions.log showing “hiccup” when running the tecSend action

4.4 Zenoss / TEC configuration using a page alert

This sample solution extends some of the ideas in the event command scenario to perform more complex event mapping between Zenoss and TEC. It uses the Zenoss Page alert mechanism to generate postmsg commands. Clearing events will be forwarded to TEC as well as “bad news” events.

On TEC, a ruleset is developed to detect duplicate events and to ensure “good news” events close “bad news” events.

4.4.1 TEC configuration

The following table shows the proposed mapping between Zenoss event fields and TEC event attributes. The fields marked with an asterisk denote new attributes that do not exist in the TEC base event.

Zenoss Event	TEC Event
eventClass	zEventClass *
evid	zEvid *
component	zComponent *
device	hostname
ipAddress	origin
summary	msg
severity	severity

Table 4.2: Mapping Zenoss event fields to TEC event attributes

In addition to mapping fields, severity definitions are different between Zenoss and TEC so the following conversion will be used:

Zenoss severity	TEC severity
Critical (5) (red)	FATAL (black)
Error (4) (orange)	CRITICAL (red)
Warn (3) (yellow)	WARNING (yellow)
Info (2) (blue)	UNKNOWN (blue)
Debug (1) (grey)	UNKNOWN (blue)
Clear (0) (green)	HARMLESS (green)

Table 4.3: Mapping Zenoss severities to TEC severities (TEC Minor severity not used)

Note that Zenoss severities are actually held in the database as numeric values. Also note that although the Zenoss Event Console lists **Warning** as a status, the Zenoss zensendevent command needs such a severity specified as **Warn**. TEC severities are defined as an enumerated type; that is, literal strings such as **WARNING**.

On the TEC Server, a new TEC class, *Zenoss_sendTec*, will be defined in *zenoss.baroc* (see Figure 15). This class defaults the **adapter_host** attribute to *zenoss.class.example.org* which means that if the TEC adapter does not populate this attribute, then the default will be applied by the reception engine of the Event Server. Three new STRING event attributes are defined:

- zEventClass for the original Zenoss class
- zEvid for the unique Zenoss event id

- zComponent for the Zenoss component field

The only event attributes with the dup_detect facet are **zEvid** and **severity**. This should be adequate since every Zenoss event should have a unique event id.

Zenoss_sendTec inherits from the base event, as did the earlier Zenoss_Base event. It is common to build hierarchies of TEC classes for a particular application, with each subclass inheriting attributes from its parent. To keep things simpler for this paper, TEC class hierarchies have not been introduced. See the “TEC Rule Developer’s Guide” for more information on class hierarchies.

```

# Base TEC class for Zenoss events

TEC_CLASS :
    Zenoss_Base ISA EVENT
    DEFINES {
        source: default= "ZENOSS";
        sub_source: dup_detect=yes;
        sub_origin: dup_detect=yes;
        adapter_host: default= "N/A";
        msg_catalog: default= "none";
        msg_index: default= 0;
        repeat_count: default= 0;
        severity: default = WARNING, dup_detect=yes;
        hostname: dup_detect=yes;
    };
END

TEC_CLASS :
    Zenoss_sendTec ISA EVENT
    DEFINES {
        source: default= "ZENOSS";
        adapter_host: default= "zenoss.class.example.org";
        msg_catalog: default= "none";
        msg_index: default= 0;
        repeat_count: default= 0;
        severity: dup_detect=yes;
        zEventClass:    STRING;
        zEvid:          STRING, dup_detect=yes;
        zComponent:    STRING;
    };
END

```

Figure 15: zenoss.baroc class file with Zenoss_sendTec definition

Although strictly, Zenoss should not execute duplicate actions, the “hiccup” described at the end of section 4.3.4 means that sometimes it does! This results in duplicate event forwarding to TEC.

The dup_detect facets on the Zenoss_sendTec class definition defines what attributes have to match, along with the TEC class, in order for TEC to consider the event a duplicate. An addition, a rule is needed to process such duplicates – see Figure 16.

The **filter_duplicate_zenoss** rule checks the event under analysis for a class of *Zenoss_sendTec*. If the class matches, the event repository database is searched for the most recent duplicate event, that is not already CLOSED, searching back for upto 10 minutes (600 seconds). If a duplicate is found, the


```

/* zenoss.baroc has dup_detect on Zenoss_sendTec for
zEvid and severity */

rule: filter_duplicate_zenoss: (
description: 'Filter duplicates for Zenoss events',
event:_event of_class 'Zenoss_sendTec',
action: filter: (
    first_duplicate(_event, event: _dup_ev
                    where [
                        status: outside ['CLOSED']
                    ],
                    _event - 600 - 600),
    add_to_repeat_count(_dup_ev, 1),
    drop_received_event
    )
).

```

Figure 16: *filter_duplicate_zenoss* rule in *zenoss.rls*

original event has its **repeat_count** field incremented and the new event is dropped.

Zenoss has an automatic clearing action whereby an event with a severity of **Clear** will automatically clear other events with the same *Zenoss eventClass*, *device* and *component* fields. TEC doesn't have the same built-in ability but it is easy enough to code a rule that does this, with the extra flexibility of having more control of exactly what “good news” event clears which “bad news” events. See the *auto_close_zenoss_bad_with_zenoss_clear* rule in Figure 17.

The incoming, event under analysis is checked for a class of *Zenoss_sendTec* and a severity of *HARMLESS*; if these conditions are met, the *zEvid* field is collected from the incoming event and a search is made through the event repository for the most recent event of **class** *Zenoss_sendTec*, with **status** not equal to *CLOSED*, with **severity** not equal to *HARMLESS*, with the same **zEvid** value as the incoming event, searching back through the event repository for upto 10 minutes.

```

rule: auto_close_zenoss_bad_with_zenoss_clear: (
description: 'Automatically close Zenoss bad news with Zenoss clear
event \
    and drop the incoming good news event \
    provided bad news event is within 10 mins (600 secs) of good news \
    matching performed on zEvid',

event: _event of_class 'Zenoss_sendTec'
    where [
        zEvid: _zEvid,
        severity: equals 'HARMLESS'
    ],

action:close: (
    first_instance(event: _down_ev of_class 'Zenoss_sendTec'
        where [
            status: outside ['CLOSED'],
            severity: outside ['HARMLESS'],
            zEvid: equals _zEvid
        ],
        _event - 600 - 600),
% Use link_effect_to_cause to be able to see closing event
% Not necessary to make logic work
    link_effect_to_cause(_down_ev, _event),

    set_event_status(_down_ev, 'CLOSED'),
    set_event_status(_event, 'CLOSED')
    )

/**/ To drop incoming HARMLESS clearing event, uncomment next lines ***/
/**/ and add comma at end of previous line after closing round bracket
***/

% action:drop: (
%     drop_received_event
%     )

).

```

Figure 17: *auto_close_zenoss_bad_with_zenoss_clear* rule in *zenoss.rls*

If such an event is found, it is linked to the incoming event. This is not necessary to make the closing logic work but does provide extra information. The historical event (pointed at by the variable *_down_ev*) is CLOSED and the incoming event (pointed at by the variable *_event*) is also CLOSED.

Check back to section 3.6 ,”TEC rulebases“ for details on incorporating *zenoss.baroc* and *zenoss.rls* into an existing rulebase and activating *it.z*

4.4.2 Zenoss configuration

Event commands can certainly be more complex than the example in the previous section; however, powerful though the filters are, it would be hard to establish different rules for forwarding to TEC (or different TECs) at different times of day.

Zenoss's alerting facility can send email or page a user. There is a single Zenoss-wide command to send an email or page and then user-specific **Alerting Rules** which can include filtering rules (exactly similar to those in event commands) and a Scheduling option to run different alerting rules at different times. By default, all alerting rules are run at all times.

Although the default paging command is configured as

```
$ZENHOME/bin/zensnpp localhost 444 $RECIPIENT
```

this could be changed to almost anything, including a command to drive postmsg. Use the *Settings* left-hand menu to configure the global *Page Command* (note that in the screenshot below, the command does not quite fit into the displayable part of the window – the actual command is *\$ZENHOME/local/sendTec.sh*).

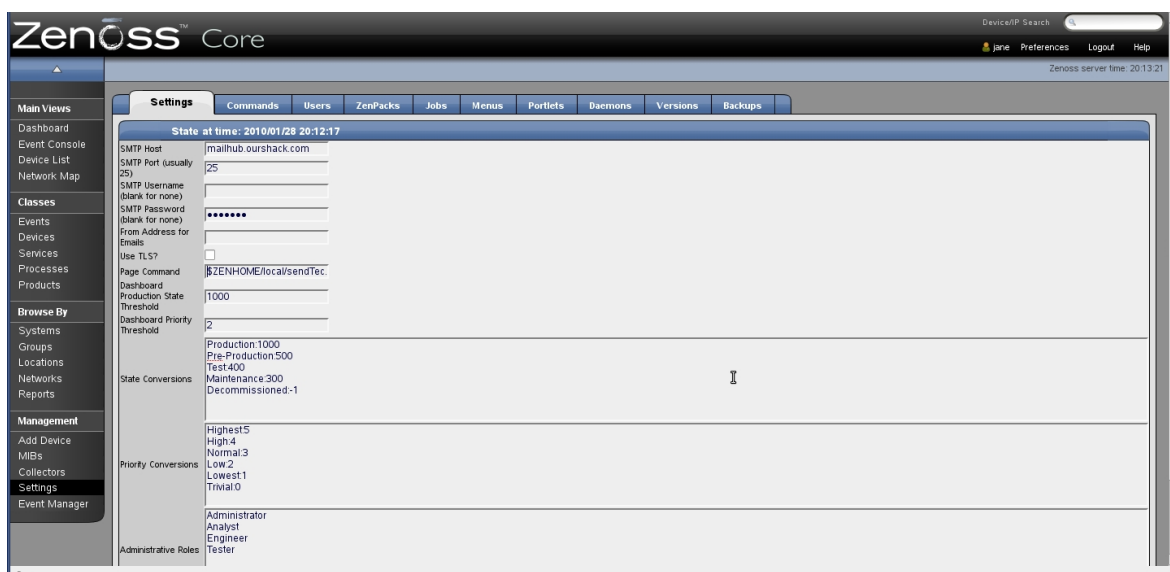


Figure 18: Configuring the global Page Command

Next, configure a Zenoss user called *tec*, using the *Users* tab.

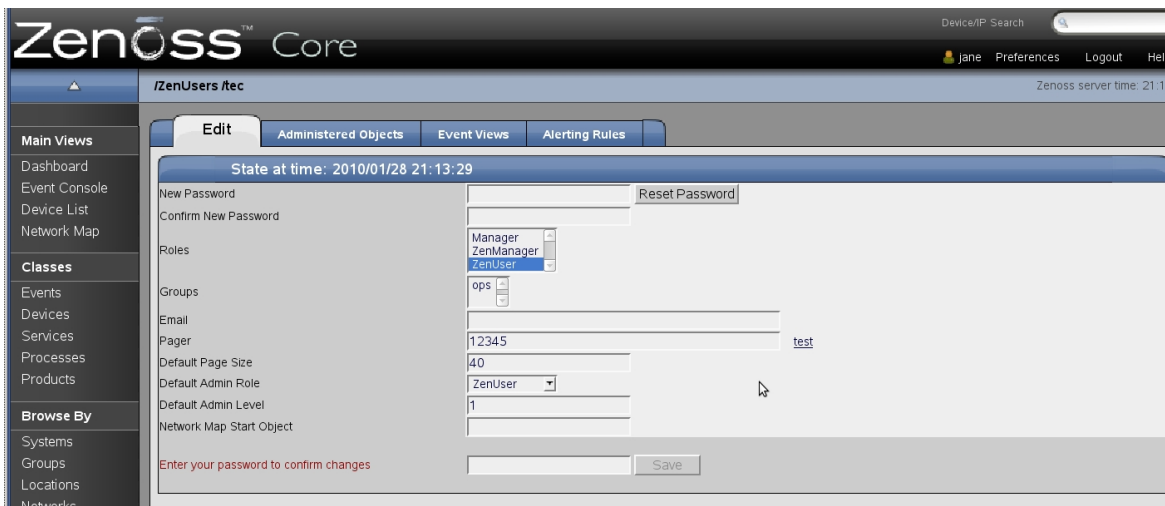


Figure 19: Basic parameters for the tec Zenoss user

Note in Figure 19 that a *Pager* value is given. Although this value is not used to send postmsg commands, leaving the field blank will result in a subsequent error. This parameter is passed to the default Page Command as the *\$RECIPIENT* variable.

The *Alerting Rules* tab provides extra flexibility, beyond what is available with event commands. To create a new Alerting Rule, use the table dropdown menu to *Add Alerting Rule* and specify a name. Once created, click on the rule name to edit it.

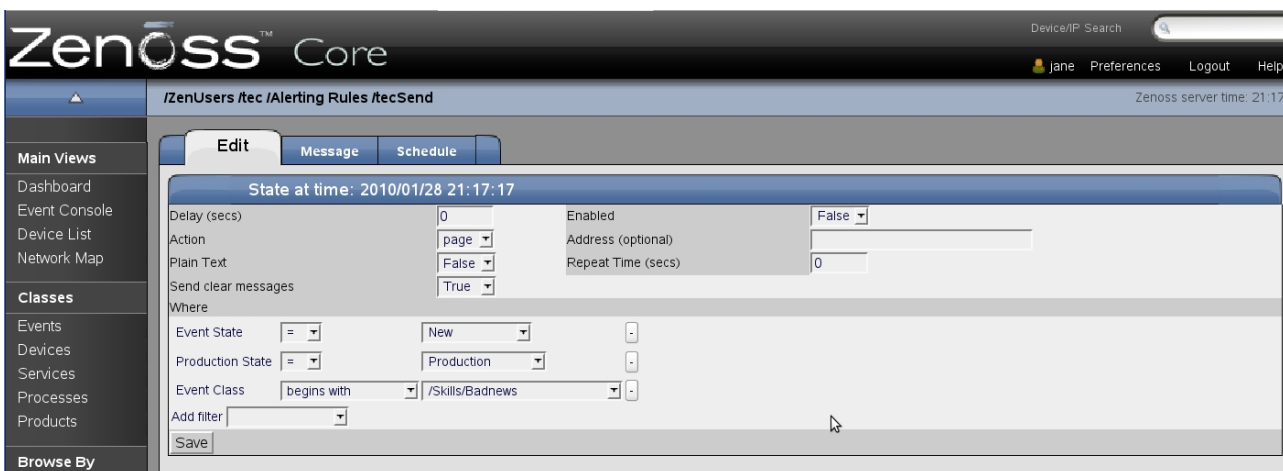


Figure 20: Basic parameters for the tec user's tecSend Alerting Rule

Note that the *Action* parameter is set to *Page* in Figure 20. This links to the global Page Command. Also make sure that the *Enabled* flag is set correctly. The bottom part of the dialogue is available for filters in exactly the same way as event commands. The alert will only be generated if all the following conditions are true

- This is a **New** event (not Acknowledged or Suppressed)
- The device that generated the event is in the **Production** state

- The eventClass begins with **/Skills/Badnews**

The detail of the Page Command is specified in the *Message* tab, which is delivered as stdin to the Page Command subshell.

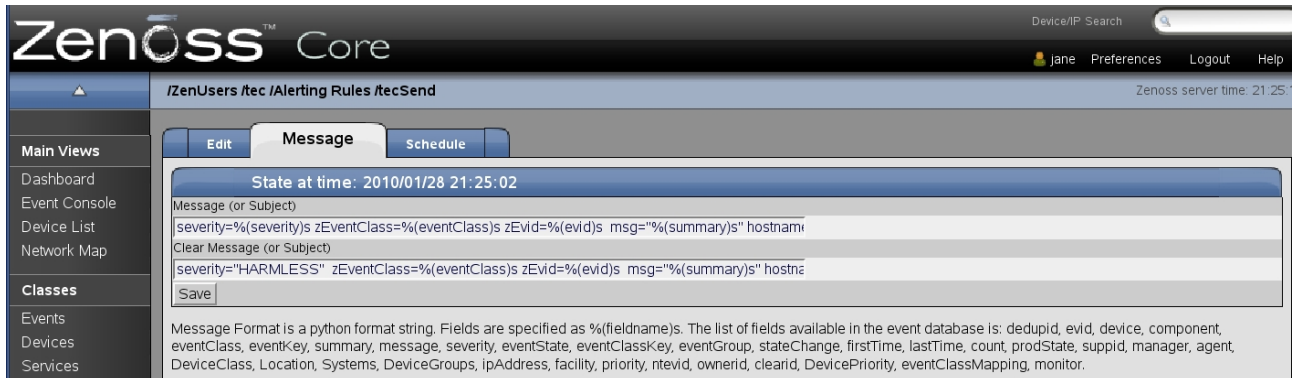


Figure 21: Message parameters sent as stdin to the page command

The complete entries are shown below (the screen truncates them in Figure 21). Each entry should all be on one line.

- **Message**

```
severity=%(severity)s zEventClass=%(eventClass)s zEvid=%(evid)s
msg="%(summary)s" hostname=%(device)s origin=%(ipAddress)s
zComponent=%(component)s
```
- **Clear Message**

```
severity="HARMLESS" zEventClass=%(eventClass)s zEvid=%(evid)s
msg="%(summary)s" hostname=%(device)s origin=%(ipAddress)s
zComponent=%(component)s
```

Note the useful help at the bottom explaining what fields from the event are available for substitution.

This dialogue does not use TALES substitution parameters like the event command; it uses Python string format. Unlike event commands, attributes of the device that generated the event are **not** available (unless they also exist as fields of the event).

This "page" alerting solution relies on using a local script, *\$ZENHOME/local/sendTec.sh*, which integrates the Message fields into a postmsg command, as shown in Figure 22.

```
#!/bin/bash
# Note tec user must have Pager command filled in with something
# It needn't be used but must exist for sendPage method in
#$ZENHOME/Products/ZenUtils/Utils.py
POSTEMSG=/usr/local/zenoss/zenoss/local/postemsg
POSTEMSG_CFG=/usr/local/zenoss/zenoss/local/tecint.conf
TEC_CLASS="Zenoss_sendTec"
TEC_SOURCE=ZENOSS
# get alert message parameters from stdin
# and convert Zenoss severities to TEC severities, ignoring case
TEC_PARAMS=`sed \
    -e 's/severity=5/severity="FATAL"/i' \
    -e 's/severity=4/severity="CRITICAL"/i' \
    -e 's/severity=3/severity="WARNING"/i' \
    -e 's/severity=2/severity="UNKNOWN"/i' \
    -e 's/severity=1/severity="UNKNOWN"/i' \
    -e 's/severity=0/severity="HARMLESS"/i'`
# Need output from this script - expected by sendPage method in
#$ZENHOME/Products/ZenUtils/Utils.py
echo OK
# Need eval on next line or quoting gets messed up and postemsg doesn't run
eval $POSTEMSG -f $POSTEMSG_CFG "$TEC_PARAMS" $TEC_CLASS $TEC_SOURCE
# Uncomment next lines for debugging
#echo $POSTEMSG -f $POSTEMSG_CFG "$TEC_PARAMS" $TEC_CLASS $TEC_SOURCE \
# > /usr/local/zenoss/zenoss/local/sendTec.out
```

Figure 22: \$ZENHOME/local/sendTec.sh to drive Zenoss Page Command

4.4.3 Testing the page solution

Before testing the Zenoss page solution, ensure that the event command solution is disabled and the page action is enabled.

Use a similar zensendevent command as in the previous section to generate test events:

```
zensendevent -d win2003.class.example.org -s Critical -k badnews -p TestComp This is bad news 310
```

Although this command does not explicitly specify eventClass, a Zenoss event class mapping populates the eventClass field with */Skills/Badnews*, based on a regular expression (Regex) that matches the summary field with:

```
This is bad news
```

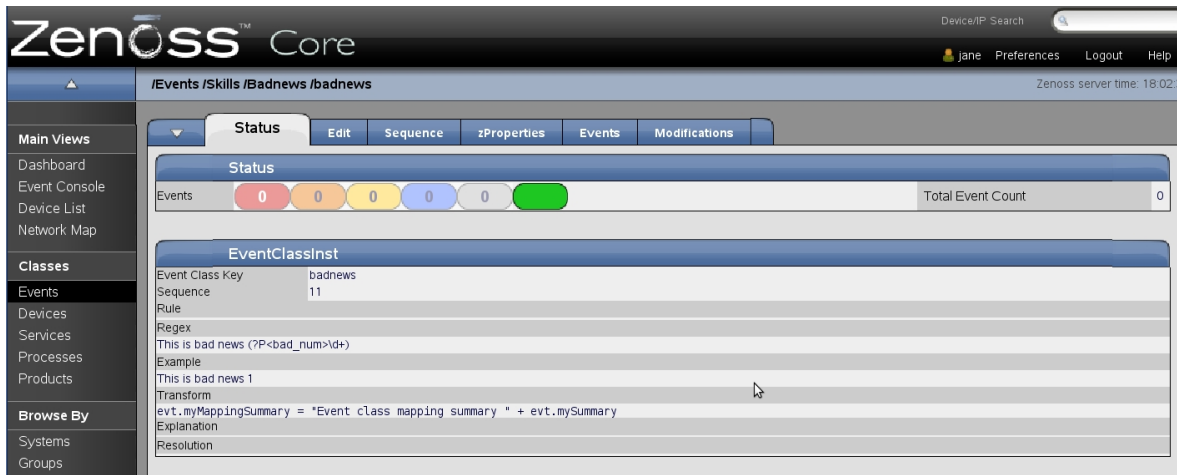


Figure 23: Zenoss event class mapping to map to eventClass /Skills/Badnews

For a much more detailed discussion on Zenoss mapping and transforms, see “Zenoss Event Management” from

http://www.skills-1st.co.uk/papers/jane/zenoss_event_management_paper.pdf.

The event should appear in the Zenoss Event Console and in the TEC Console. Check that the field / attribute mapping has worked correctly.

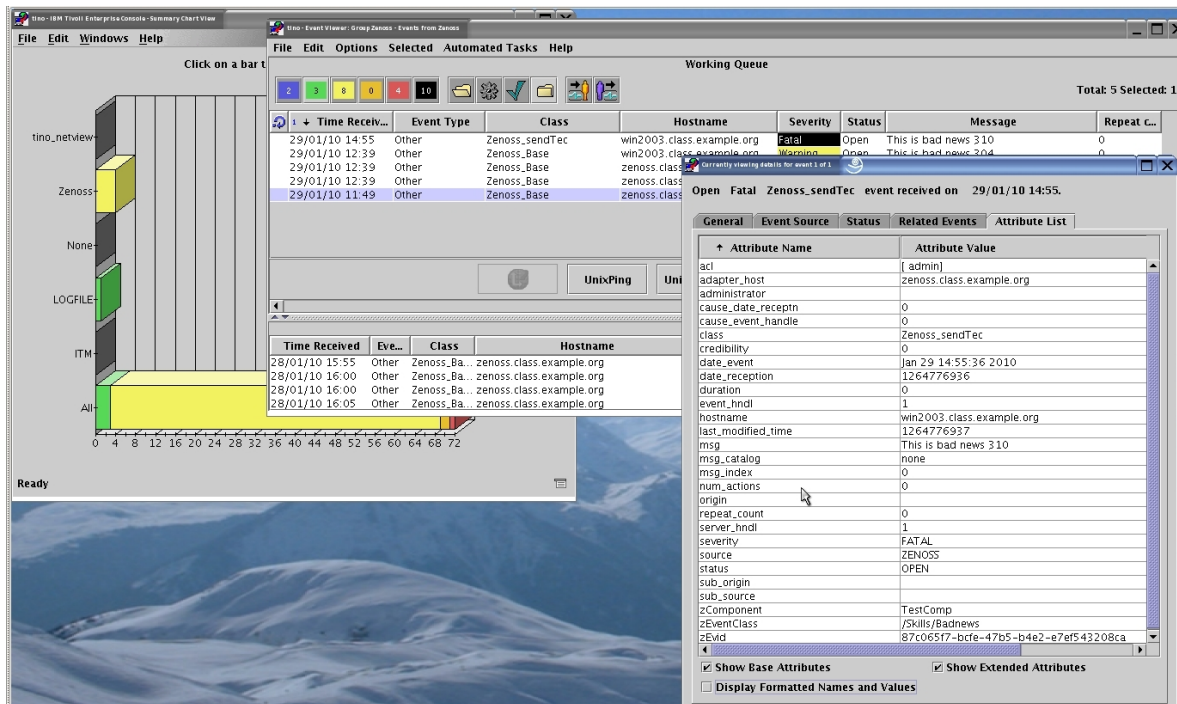


Figure 24: Zenoss_sendTec event forwarded to TEC Console with event detail showing attribute mapping

If an exact duplicate is sent, the repeat count of the initial event should be incremented in the Zenoss Console; nothing should change at TEC as

zenactions will detect that this is a duplicate event and not execute any event commands or alerts.

To try and provoke the “hiccup” in zenactions processing that generates duplicate actions, try sending three zensendevent commands in quick succession, which only differ in the final number (I sent 311, 312 and 313). Check `$ZENHOME/log/zenactions.log` to see whether a problem has occurred (refer back to Figure 14). If the “hiccup” happens (it doesn't always), you will probably see at least one of the events in TEC with a repeat_count of 1, rather than 0.

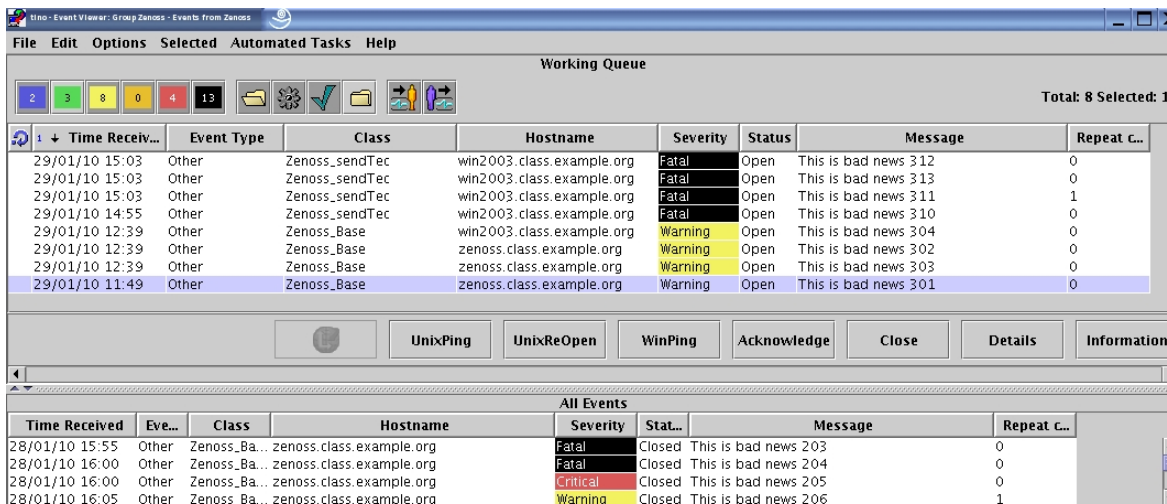


Figure 25: TEC Console showing 3 further Zenoss_sendTec events; “bad news 311” has a duplicate

To clear the events, use the following zensendevent command:

```
zensendevent -d win2003.class.example.org -s Clear -k badnews -p TestComp This is bad news 313
```

It doesn't matter what the final number is as Zenoss will automatically close all events with the same class, device and component. If you need greater control over the clearing mechanism then you need to configure Zenoss to more specifically define the event class and/or component.

All the “bad news” events should disappear from the Zenoss Events Console; they should be viewed in the Event History, along with the clearing “good news” event.

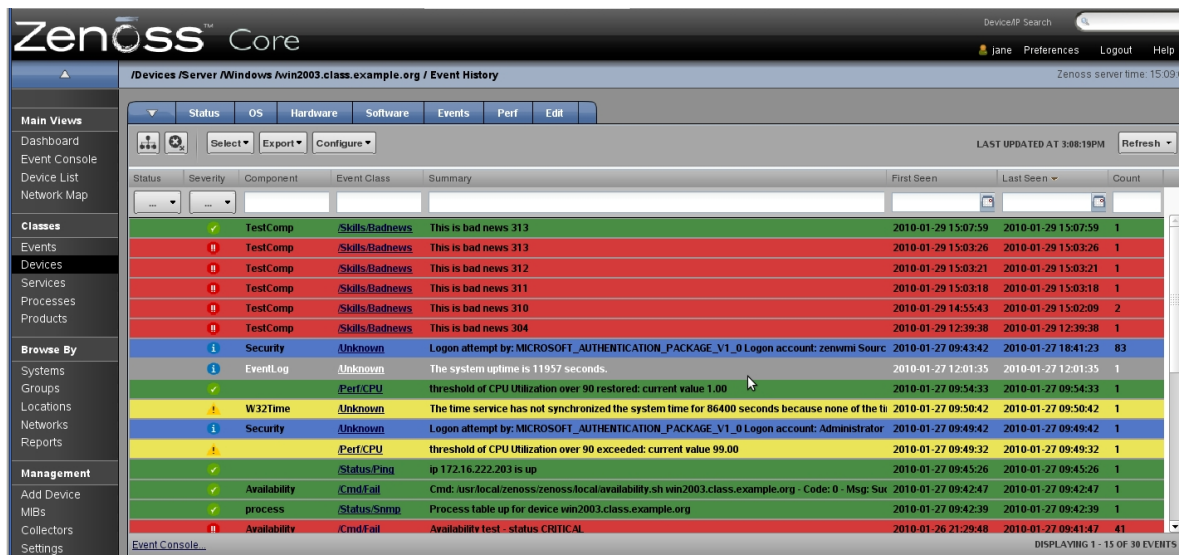


Figure 26: Zenoss Event History console showing clearing event and the cleared "bad news" events

At TEC, similarly the "bad news" events should be closed.

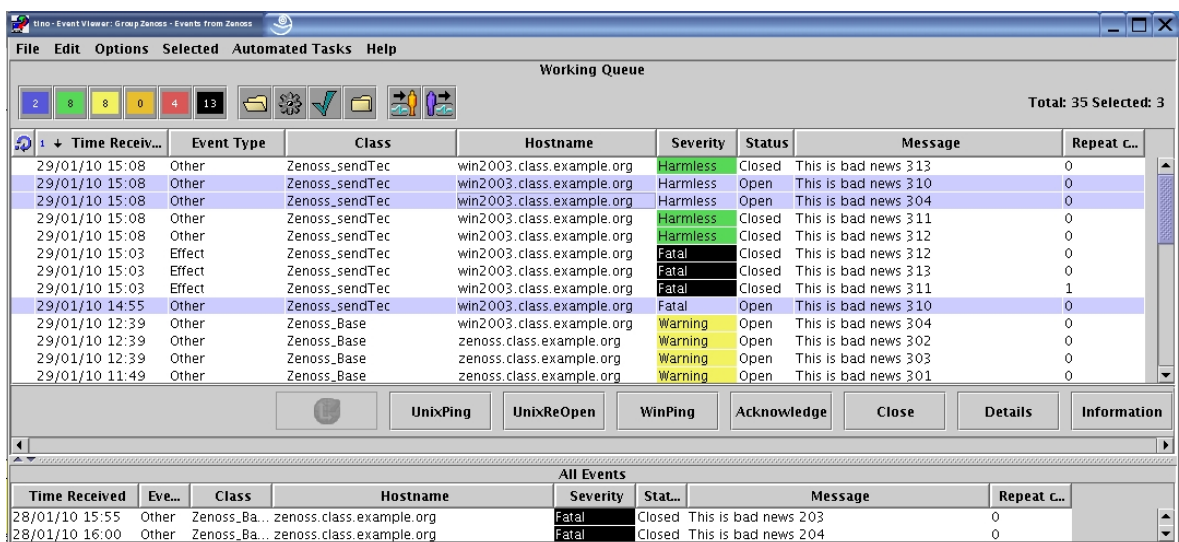


Figure 27: TEC Console showing Closed "good news" event and Closed "bad news" events

Note that there are three events that are still Open. Why has "bad news 310" not closed? Remember that the closing TEC rule only checked for previous events upto 10 minutes old. The closing event for "310" arrived 13 minutes after the bad news so the rule was not applied so neither the "good news" nor the "bad news" events are closed.

Inspect `$ZENHOME/log/zenactions.log` to see that a page action did actually take place for each closing event.

```
jane@zenoss:~/de/zenoss/zenpacks - Shell - Konsole
Session Edit View Bookmarks Settings Help

2010-01-29 15:04:58,862 INFO zen.ZenActions: processed 1 rules in 0.35 secs
2010-01-29 15:05:58,889 INFO zen.ZenActions: Processed 3 commands in 0.018029
2010-01-29 15:05:58,902 INFO zen.ZenActions: processed 1 rules in 0.04 secs
2010-01-29 15:06:58,924 INFO zen.ZenActions: Processed 3 commands in 0.014943
2010-01-29 15:06:58,937 INFO zen.ZenActions: processed 1 rules in 0.03 secs
2010-01-29 15:07:59,026 INFO zen.ZenActions: Processed 3 commands in 0.019227
2010-01-29 15:07:59,039 INFO zen.ZenActions: processed 1 rules in 0.04 secs
2010-01-29 15:08:59,192 INFO zen.ZenActions: Running
2010-01-29 15:08:59,255 INFO zen.ZenActions: Running
2010-01-29 15:08:59,321 INFO zen.ZenActions: Running
2010-01-29 15:08:59,367 INFO zen.ZenActions: Running
2010-01-29 15:08:59,431 INFO zen.ZenActions: Running
2010-01-29 15:08:59,468 INFO zen.ZenActions: Processed 3 commands in 0.401898
2010-01-29 15:08:59,508 INFO zen.ZenActions: sent page to 12345: severity="HARMLESS" zEventClass=/Skills/Badnews zEvid=5320ecc-
71e5-4409-ad86-bf73bb221873 msg="This is bad news 312" hostname=win2003.class.example.org origin= zComponent=TestComp
2010-01-29 15:08:59,548 INFO zen.ZenActions: sent page to 12345: severity="HARMLESS" zEventClass=/Skills/Badnews zEvid=694f35c8-
8d59-4d66-9ce8-0801422e792f msg="This is bad news 313" hostname=win2003.class.example.org origin= zComponent=TestComp
2010-01-29 15:08:59,589 INFO zen.ZenActions: sent page to 12345: severity="HARMLESS" zEventClass=/Skills/Badnews zEvid=87c065f7-
bcfe-47b5-b4e2-e7ef543208ca msg="This is bad news 310" hostname=win2003.class.example.org origin= zComponent=TestComp
2010-01-29 15:08:59,639 INFO zen.ZenActions: sent page to 12345: severity="HARMLESS" zEventClass=/Skills/Badnews zEvid=c6518025-
ed15-4f51-8002-368605f91f6e msg="This is bad news 304" hostname=win2003.class.example.org origin= zComponent=TestComp
2010-01-29 15:08:59,678 INFO zen.ZenActions: sent page to 12345: severity="HARMLESS" zEventClass=/Skills/Badnews zEvid=dd5bbad9-
1c77-421e-9f98-755805ff6658 msg="This is bad news 311" hostname=win2003.class.example.org origin= zComponent=TestComp
2010-01-29 15:08:59,682 INFO zen.ZenActions: processed 1 rules in 0.62 secs
2010-01-29 15:09:59,705 INFO zen.ZenActions: Processed 3 commands in 0.015856
2010-01-29 15:09:59,716 INFO zen.ZenActions: processed 1 rules in 0.03 secs
```

Figure 28: zenactions.log showing alert actions for closing events

5 Conclusions

The Zenoss events subsystem and the IBM Tivoli Enterprise Console (TEC) have many similarities although the skills required to configure each are very different. TEC is fundamentally a Prolog engine; Zenoss is fundamentally a Python engine.

Ultimately, TEC may scale better than the open source Zenoss Core although the chargeable Zenoss Enterprise offering has extra distributed architecture so should scale better than Core. Zenoss Core is known to effectively manage enterprises with well over a thousand devices. The assumption in this paper is that TEC will be the higher-level manager with Zenoss feeding in to it.

Two integration techniques are examined for Zenoss:

- Event Commands
- Page Alerts

They are summarised in the table below. They both use the TEC `postmsg` command driven by Zenoss's `zenactions` daemon. Setup effort for either solution is similar.

Event Commands	Page Alerts
Shellscript run by <code>zenactions</code>	Shellscript run by <code>zenactions</code>
Parameters passed as TALES expressions	Parameters passed in Python string format
Parameters include event fields and device attributes	Parameters only include event fields
No time criteria for running commands	Schedule tab for Alerting Rules to control what runs when
	Alerting Rules are per user – separate users could be defined if there are several upstream TEC Servers
Extensive filtering capability to control what events generate commands	Extensive filtering capability to control what events generate commands

Table 5.1: Similarities and differences between event commands and page alerts

Some TEC configuration will be required although this can be kept fairly minimal if only simple event forwarding is required. The more complete the integration solution, then the more work will be required, especially on TEC baroc class files and TEC rules files.

Although this paper only discusses integrating Zenoss with TEC, IBM's newer problem management offering, Netcool/OMNIbus, can also accept events in

TEC format through its EIF probe; hence the solutions shown here could also be used to integrate between Zenoss and Netcool/OMNIbus.

Indeed, these general Zenoss techniques could be used to integrate Zenoss into **any** other problem management system, provided it is possible to format events for the upstream manager within a shellsript.

If organisations require two-way integration between Zenoss and TEC, for example where events are closed in TEC and this change should be reflected back to Zenoss, this might be achieved by using a TEC rule to run a script that generates a specific SNMP TRAP to the Zenoss server. A Zenoss event mapping could interpret that TRAP and run an action that generates a clearing zensendevent, with appropriate substituted parameters.

To conclude, both Zenoss and TEC are powerful event management systems in their own right; together they can deliver even greater returns.

References

1. Zenoss network, systems and application monitoring - <http://www.zenoss.com/>
2. Zenoss community website - <http://community.zenoss.org/>
3. Zenoss Administration Guide - <http://community.zenoss.org/community/documentation>
4. Zenoss Developer's Guide - <http://community.zenoss.org/community/documentation>
5. “Zenoss Core Network and System Monitoring” by Michael Badger, published by PACKT Publishing, June 2008, ISBN 978-1-847194-28-2 .
6. For information on TALES expressions, see http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/AppendixC.stx
7. For documentation on Zenoss functionality, the ZEO object database and Zope. Try: <http://www.zenoss.com/community/docs/zenoss-api-docs/2.1/>
8. As a general Python reference, try “Learning Python” by Mark Lutz, published by O'Reilly
9. For detailed information on Zenoss's event architecture, get “Zenoss Event Management” from http://www.skills-1st.co.uk/papers/jane/zenoss_event_management_paper.pdf .
10. Consult http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.itec.doc_3.9/toc.xml for TEC online manuals.

About the author

Jane Curry has been a network and systems management technical consultant and trainer for 25 years. During her 11 years working for IBM she fulfilled both pre-sales and consultancy roles spanning the full range of IBM's SystemView products prior to 1996 and then, when IBM bought Tivoli, she specialised in the systems management products of Distributed Monitoring & IBM Tivoli Monitoring (ITM), the network management product, Tivoli NetView and the problem management product Tivoli Enterprise Console (TEC). All are based around the Tivoli Framework architecture.

Since 1997 Jane has been an independent businesswoman working with many companies, both large and small, commercial and public sector, delivering Tivoli consultancy and training. Over the last 5 years her work has been more involved with Open Source offerings. She was made a Zenoss Master by Zenoss in February 2009.