

**Integrating
IBM Operations Manager for z/VM®
with an Event Management System such
as
IBM Tivoli Netcool®/OMNIbus**



June 1, 2019

**Tracy Dean
IBM Product Manager
IBM z/VM Management Software**

Special Notices

This document reflects the IBM Advanced Technical Skills understanding of many of the questions asked about the integration of Operations Manager for z/VM and Netcool/OMNIBus. It was produced and reviewed by the members of the IBM Advanced Technical Skills organization. This document is presented “As-Is” and IBM does not assume responsibility for the statements expressed herein. It reflects the opinions of the IBM Advanced Technical Skills. These opinions are based on hands on experiences with IBM Operations Manager for z/VM, Event Integration Facility, and Netcool/OMNIBus. If you have questions about the contents of this document, please direct them to Tracy Dean.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries: IBM, Netcool, System z, z/VM. A full list of U.S. trademarks owned by IBM may be found at <http://www.ibm.com/legal/copytrade.shtml>.

Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

LINUX is a registered trademark of Linus Torvalds.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Other company, product and service names may be trademarks or service marks of others.

Acknowledgements

Many thanks to Mike Bonett and Mike Sine for their work to create the initial white paper. Thanks to Art Eisenhour, IBM Advanced Technical Skills, for reviewing this paper and providing input.

Introduction	4
Required Components	4
EIF event from z/VM to Netcool/OMNIbus	7
Operations Manager Configuration	7
Event Integration Facility (EIF) Configuration	8
Tivoli EIF Probe Configuration	10
Netcool/OMNIbus Configuration	11
EIF Event Flow Example	12
Event Flow Example	12
SNMP Trap from z/VM to Netcool/OMNIbus	15
z/VM SNMP Customization	15
Creating SNMP Traps	16
Operations Manager Configuration	17
SNMP Probe	18
Netcool/OMNIbus Configuration	20
SNMP Trap Flow Example	21
Summary	24

Introduction

IBM Operations Manager for z/VM provides an automation solution for the z/VM environment, including Linux guests. One of its capabilities is to execute commands on Operations Manager servers or on Linux guests, based on an event on the z/VM system or a Linux guest. With this function, it is possible to have Operations Manager send events to Netcool/OMNIBus, and integrate events detected in the z/VM environment with other events across the enterprise. This white paper covers two methods of integrating Operations Manager with Netcool/OMNIBus:

- Using the Tivoli Event Integration Facility (EIF) via the POSTZMSG or POSTEIFMSG command in Linux
- Using Simple Network Management Protocol (SNMP) via the SNMPTRAP command in z/VM. This method can be used to integrate with Netcool/OMNIBus and many other event management systems.

It will explain what must be configured to provide this capability for each method and walk through execution examples.

Required Components

- **IBM Operations Manager for z/VM**
Operations Manager for z/VM is an IBM product that provides several functions designed to improve productivity of z/VM system programmers and operations staff. Consoles of z/VM user IDs (such as z/VM service machines, Linux guest consoles, and Linux SYSLOG data) can be monitored for specific messages. Actions can be defined to automatically respond to certain messages, resolving the issue without operator interference. Authorized users can view and interact with these consoles from their own user ID, without logging on to the Linux guest or z/VM service machine. A scheduling function is also included, allowing you to automatically execute an action at certain times of the day or on specific days of the week. Monitors can be defined to automatically respond to spool or page space conditions, such as a spool or page space approaching full or rapidly increasing use of spool or page space. Authorized users can search for and view spool files that meet specified criteria based on the file owner, file size, or the date the file was created. Monitors can also be defined for various system events, such as a user ID or guest logging off and entering the CP Read state, or a member leaving or joining an SSI cluster.
- **IBM Tivoli Netcool/OMNIBus**
Netcool/OMNIBus is an event management platform that supports event consolidation, correlation, and automation actions. Events can be received from a variety of sources, including EIF events and SNMP traps, to support end-to-end event management. Netcool/OMNIBus is also a component of other management products such as IBM Tivoli Network Manager and IBM Tivoli Business Service Manager. Netcool/OMNIBus is supported on various Windows, UNIX, and Linux platforms, including Linux on z Systems. Version 7.x or later of Netcool/OMNIBus is required.
- **IBM Tivoli Event Integration Facility (using the EIF method)**
The Event Integration Facility (EIF) is a component provided with Netcool/OMNIBus. It is an interface that applications can use to send or receive EIF events (these events are in the same format as Tivoli Enterprise Console (TEC) events, but are generically referred to as EIF events). The interface can be used in programming languages, or via a command line. The EIF is documented in the *IBM Tivoli*

Netcool/OMNIBus Integration Reference (SC23-9686). The postzmsg or posteifmsg from the EIF is used. Both function in the same manner; postzmsg is the older name, posteifmsg is newer name used in the EIF that ships with Netcool/OMNIBus. This document will refer to both interchangeably.

- **IBM Tivoli EIF Probe (using the EIF method)**

The EIF Probe is an optional component of Netcool/OMNIBus. It receives EIF events and maps them to the Netcool/OMNIBus ObjectServer event store (the alerts.status table). The EIF Probe provides a default mapping of EIF event contents to Netcool/OMNIBus, which can be easily modified if desired. The EIF Probe is supported on various Windows, UNIX, and Linux platforms, including Linux on z Systems.

- **z/VM Simple Network Management Protocol (SNMP) customization (using the SNMP method)**

SNMP is part of the z/VM TCP/IP function, but must be enabled to send SNMP traps. Traps are event notifications within the SNMP protocol, and are forwarded to a SNMP trap manager, usually network monitoring software or event management systems. In this case, traps are forwarded to the Netcool/OMNIBus SNMP Probe.

- **Netcool/OMNIBus SNMP Probe (using the SNMP method)**

The SNMP Probe is an optional component of Netcool/OMNIBus. It receives SNMP traps, and maps them to the Netcool/OMNIBus ObjectServer event store. OMNIBus can read SNMP management information base files (MIBs) to determine the trap fields for predefined traps; the probe rules can also be customized so that any trap field can be mapped to any OMNIBus event field. The SNMP Probe is supported on various Windows, UNIX, and Linux platforms, including Linux on z Systems.

Proper configuration of each component is required to successfully support event flows from Operations Manager to Netcool/OMNIBus.

How to Choose Between the EIF and SNMP Methods

While both the EIF method and the SNMP method will work for Netcool/OMNIbus, each has its own advantages and disadvantages:

EIF method:

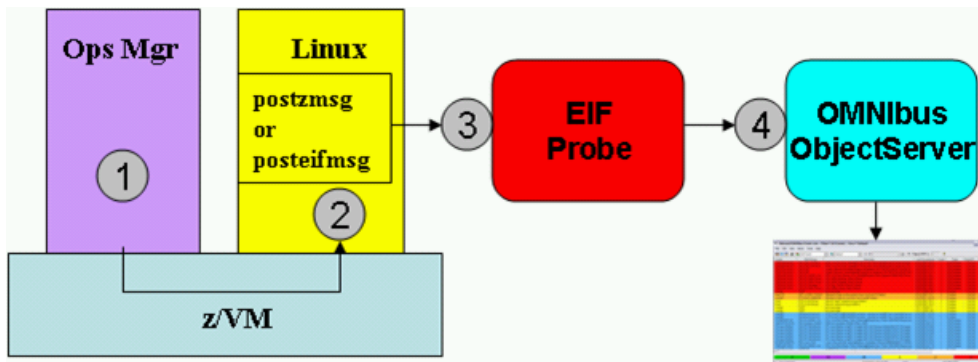
- Advantage
 - You can direct the alert to only the IP address(es) you specify
- Disadvantages
 - You need a Linux guest running and logged on that can run POSTZMSG. It must be on a z/VM system that has Operations Manager. The system must have IP connectivity to the system where the issue upon which you want to act occurs.
 - There is a limit of 160 characters on POSTZMSG command sent to the Linux guest (since Operations Manager sends it using CP SEND). This means you can't always include the full text of error message which triggered the action.

SNMP method:

- Advantages
 - There is no requirement for a Linux guest. SNMP runs on z/VM.
 - There is no limit on the message size.
- Disadvantage
 - All SNMP alerts on z/VM go the same set of IP addresses you have configured for the SNMP server on z/VM. You can not send different SNMP alerts to different targets.

EIF event from z/VM to Netcool/OMNIBus

The general process flow for using the EIF method is shown in the following diagram:



1. Based on a detected situation, Operations Manager creates an EIF command with the desired information and sends it to a Linux on System z guest that has the Event Integration Facility installed.
2. The Linux guest executes the EIF command to create the event and uses the Event Integration Facility **postzmsg** or **posteifmsg** command to send it to the EIF Probe.
3. The EIF Probe receives the event and applies its rules to map the event contents to the Netcool/OMNIBus alerts.status table.
4. Netcool/OMNIBus stores the event for use in its event management functions (display, correlate, trigger automation, etc).

Operations Manager Configuration

Rules for detecting information in monitored virtual machine consoles and taking associated actions are defined in an Operations Manager configuration file:

```
* Send an alert to Netcool/OMNIBus for abends on consoles
DEFRULE NAME ABNDOMNI +
  MATCH '*abend*' +
  ACTION ALRTOMNI

* Call POSTZMSG EXEC on z/VM to send alert to Netcool/OMNIBus
DEFACTN NAME ALRTOMNI +
  COMMAND 'EXEC POSTZMSG &u' +
  ENV LVM
```

In the above example, the Operations Manager configuration file statements will send an alert to Netcool/OMNIBus if the word “abend” is detected on any monitored console. Many other options are available, such as limiting which consoles are included in the rule, which text is included or excluded from the rule, etc.

- The DEFRULE statement defines the information to be looked for on the monitored consoles, and the action to be taken (ALRTOMNI) when that information appears.
- The DEFACTN statement defines the ALRTOMNI action, which invokes an EXEC called POSTZMSG. The &u parameter associated with the command EXEC POSTZMSG is an Operations Manager predefined substitution variable for the DEFACTN command and substitutes the user ID of the message text originator.

Following is a sample POSTZMSG EXEC (in REXX) that runs on z/VM and sends a command to the Linux guest (**postzmsg**, which will be covered in the next section). In this example, the name of the Linux guest is LNX001. This EXEC must be accessible to the Operations Manager server (OPMGRM1) and all action processing servers (OPMGRSn).

```
/* Call POSTZMSG command on Linux guest to send */
/* alert to Netcool/OMNIBus */

Trace 0
Address Command
Parse Arg baduser .

cmdpart1 = './postzmsg -f e2o.conf -r CRITICAL'
cmdpart2 = '-m guest_is_abending hostname=baduser'
cmdpart3 = 'sub_source=postzmsg origin=baduser'
cmdpart4 = 'sub_origin=tcp WARN_EVENT OpsMgr'

'CP SEND LNX001' cmdpart1 cmdpart2 cmdpart3 cmdpart4

Exit
```

Event Integration Facility (EIF) Configuration

The EIF provides several command line executables that generate EIF events. For this process the **postzmsg** command is used. In versions of the EIF that ship with Netcool/OMNIBus 7.x, the command is called **posteifmsg**, but the functions are the same. It is supported on Linux on System z, and must be installed on a Linux guest on the same z/VM system as Operations Manager. The command has the following syntax:

```
postzmsg { -S <server> | -f <config_file> } [-r <severity>]
[-m <message> ] [<slot_name=value>, ...] <class> <source>
```

For this integration process, the following parameters are needed:

- **-f <config_file>** : <config file> is a text file containing, at a minimum, the following two parameters:
 - ServerLocation: the hostname or IP address where the event will be sent. In this case it is where the EIF Probe is running.
 - ServerPort: the IP port where the event receiver (in this case the EIF Probe) is waiting to receive events.

There are other optional parameters that control the connection state, event caching and buffer, etc. that can be used (and are documented in the EIF manual).

- **-r <severity>**: This sets the event severity. It is mapped to the severity slot name for use by the EIF Probe. It must be one of the following strings (and must be upper case):
 - FATAL
 - CRITICAL
 - MINOR
 - WARNING
 - HARMLESS
 - UNKNOWN

The EIF Probe default rules map each of these to a Netcool/OMNIBus event severity.

- **-m <message>**: the message string describing the event. If it contains spaces, it must be enclosed in double quotes. The string gets mapped to the msg slot name for use by the EIF Probe.
- **slot_name=value,...**: a set of name-value pairs conveying information in the event. The slot names must match the names expected by the Tivoli EIF Probe, for the corresponding value to be mapped to the desired Netcool/OMNIBus event attribute(s). Names that are not used in the EIF Probe rules are ignored, and their associated values are not included in the Netcool/OMNIBus event. At a minimum, the following slot names should be in the event:
 - hostname
 - sub_source
 - origin
 - sub_origin
 - status
- **class**: this is the event class of the event. It is more meaningful when the event is sent to a TEC server. This parameter is mapped to the **event_class** slot name for use by the EIF Probe.
- **source**: this is the source of the event. It is more meaningful when the event is sent to a TEC server, this parameter is mapped to the **source** slot name for use by the EIF Probe.

For example, assume that the z/VM user ID OPMGRC1 displayed an abend message. When the following command string sent from Operations Manager is executed:

```
postzmsg -f e2o.conf -r CRITICAL -m guest_is_abending hostname=OPMGRC1
sub_source=postzmsg origin=OPMGRC1
sub_origin=tcp WARN_EVENT OpsMgr
```

The slot name mapping sent to the EIF Probe will be:

Name	Value
event_class	WARN_EVENT
source	OpsMgr
sub_source	postzmsg
origin	OPMGRC1
sub_origin	tcp
hostname	OPMGRC1
severity	CRITICAL
msg	guest_is_abending

Tivoli EIF Probe Configuration

The EIF Probe runs as a process on UNIX/Linux environments or a service on Windows environments. It does not have to be on the same platform as the Netcool/OMNIBus ObjectServer, but network issues between the EIF Probe and Netcool/OMNIBus will be minimized if it is.

After installing the EIF Probe, two files must be configured:

- **tivoli_eif.props:** This is the properties file for the EIF Probe. The port it listens on and the target Netcool/OMNIBus server are defined in this file.
- **tivoli_eif.rules:** This contains the rules for how the events received by the EIF Probe will be processed and mapped to Netcool/OMNIBus attributes before the event is sent to the Netcool/OMNIBus ObjectServer. The rules file syntax is documented in *IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide (SC23-6373)*.

A default set of rules are provided by the EIF Probe for matching EIF event slot names to Netcool/OMNIBus event attributes:

EIF Event slot name or constants	OMNIBus Attribute mapped to
hostname + source + sub_source + sub_origin + event_class	Identifier
source + sub_source + sub_origin	AlertKey
event_class	AlertGroup
Msg	Summary
Origin	Node
Origin	NodeAlias
“tivoli EIF probe on ”+hostname()	Manager
Source	Agent
6601	Class
severity FATAL / 60 = Critical/5 CRITICAL / 50 = Critical/5 MINOR / 40 = Minor/3 WARNING / 30 = Warning/2 UNKNOWN / 10 = Indeterminate/1	Severity
Status	TECStatus
Getdate	LastOccurrence/ FirstOccurrence
Date	TECDate
server_handle (extracted from last element of server_path list)	TECServerHandle
event_handle (extracted from last element of server_path list)	TECEventHandle
date_reception (extracted from last element of server_path list)	TECDateReception

EIF Event slot name or constants	OMNibus Attribute mapped to
repeat_count	TECRepeatCount
Fqhostname	TECFQHostname
Hostname	TECHostname

Any other EIF event slot names are ignored (they can be used by customizing the default rules).

The tivoli_eif.rules file can be customized to alter the mapping (include other slot names, change the default mapping, etc.). A custom file can be included in the tivoli_eif.rules file as the last statement, and anything in this file will override the earlier mappings.

Netcool/OMNibus Configuration

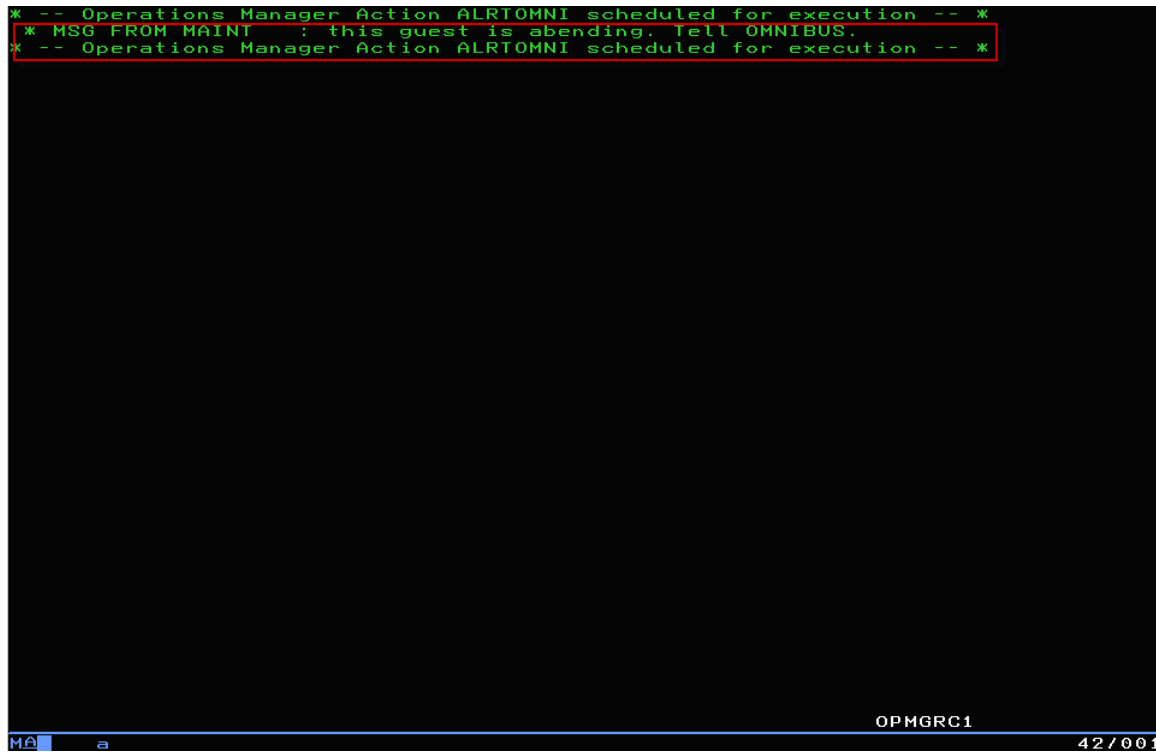
The Netcool/OMNibus server alerts.status table contains the Netcool/OMNibus events. The fields in this table are the target of the EIF Probe EIF event mapping. To support this mapping the default alerts.status table is extended by adding additional fields. The files used to extend this mapping (sql statements executed against the Netcool/OMNibus ObjectServer via the nco_sql command on UNIX/Linux or the isql command on Windows) are provided as part of the EIF Probe, or can be obtained from the IBM Tivoli Open Process Automation Library (OPAL) website at <http://www.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10EC01>.

The ability of the Netcool/OMNibus ObjectServer to support EIF integration can be validated by viewing the columns in the alerts.status table. If columns beginning with TEC_ exist, then the required updates have been applied. If not, the update will have to be applied (instructions are provided with the EIF Probe or in the package provided at the OPAL website link above).

EIF Event Flow Example

This example will walk through how the components tie together to send an EIF event from z/VM to Netcool/OMNIBus.

1. Operations Manager detects an incident for which an event will be sent.



The screenshot shows a z/VM console window with a black background and green text. At the top, there is a message from the Operations Manager. The message is enclosed in a red rectangular box. The text of the message is as follows:

```
* -- Operations Manager Action ALRTOMNI scheduled for execution -- *  
* MSG FROM MAINT : this guest is abending. Tell OMNIBUS. *  
* -- Operations Manager Action ALRTOMNI scheduled for execution -- *
```

Below the message, the rest of the console is black. At the bottom of the console, there is a status bar with the text "OPMGRC1" on the left and "42 / 001" on the right.

2. Operations Manager DEFACTN statement invokes a REXX EXEC that sends the postzmsg command to a Linux guest that has the EIF installed. The following screen capture shows a view of the Linux guest console, using Operations Manager:

```

A - DEMOADMN.ATS
File Edit View Communication Actions Window Help
23:59:59
11:24:47 cd /workloads
11:24:47 hasl112:/workloads # ./postzmsg -f e2o.conf -r CRITICAL -m guest_is_ab
11:24:47 hasl112:/workloads #
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT FRIDAY 04/24/09
23:59:59
11:26:55 cd /workloads
11:26:55 hasl112:/workloads # ./postzmsg -f e2o.conf -r WARNING -m fatal_error_
11:26:55 hasl112:/workloads #
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT SATURDAY 04/25/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT SUNDAY 04/26/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT MONDAY 04/27/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT TUESDAY 04/28/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT WEDNESDAY 04/29/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT THURSDAY 04/30/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT FRIDAY 05/01/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT SATURDAY 05/02/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT SUNDAY 05/03/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT MONDAY 05/04/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT TUESDAY 05/05/09
23:59:59
13:12:01 cd /workloads
13:12:01 hasl112:/workloads # ./postzmsg -f e2o.conf -r CRITICAL -m guest_is_ab
13:12:01 hasl112:/workloads #
13:46:59 cd /workloads
13:46:59 hasl112:/workloads # ./postzmsg -f e2o.conf -r CRITICAL -m guest_is_ab
13:46:59 hasl112:/workloads #
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT WEDNESDAY 05/06/09
23:59:59
23:59:59 HCPMID6001I TIME IS 00:00:00 EDT THURSDAY 05/07/09
23:59:59
-
ESMTS112 (Scroll)
MA a 42/001

```

- The Linux guest executes the command and sends the following event string to the EIF Probe, using the e2o.conf file indicated by the -f parameter to determine the EIF Probe location:

```

SCARY_EVENT;source=OpsMgr;severity=CRITICAL;hostname=OPMGRC1;
sub_source=postzmsg;sub_origin=tcp;msg='guest_is_abending';
origin=OPMGRC1;END

```

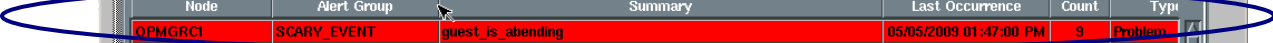
- The EIF Probe processes the received event and sends it to the Netcool/OMNIbus ObjectServer. The following is seen in the EIF Probe log:

```

Debug: D-UNK-000-000: [Event Processor] sub_source: postzmsg
Debug: D-UNK-000-000: [Event Processor] hostname: OPMGRC1
Debug: D-UNK-000-000: [Event Processor] origin: OPMGRC1
Debug: D-UNK-000-000: [Event Processor] severity: CRITICAL
Debug: D-UNK-000-000: [Event Processor] sub_origin: tcp
Debug: D-UNK-000-000: [Event Processor] source: OpsMgr
Debug: D-UNK-000-000: [Event Processor] msg: guest_is_abending
Debug: D-UNK-000-000: [Event Processor] EventSeqNo: 1
Debug: D-UNK-000-000: [Event Processor] Processing alert {0 remaining}
Debug: D-UNK-000-000: Default @Severity set to := [5] just after case statement in common
section on tivoli_eif.rules.
Debug: D-UNK-000-000: Flushing events to ObjectServer

```

- The event can now be seen on the ObjectServer console (and further event correlation/automation applied if desired):



Node	Alert Group	Summary	Last Occurrence	Count	Type
OPMGRC1	SCARY_EVENT	guest_is_abending	05/05/2009 01:47:00 PM	9	Problem
hasl112	PROBLEM_EVENT	Problem has occurred	05/05/2009 01:12:02 PM	3	Problem
T42B:CMS	ITM_ManagedSystem	MS_Offline([Status="N" AND Reason<>"FA"] ON T42B:CMS (Sta	05/01/2009 11:02:42 PM	1	ITM Probl
WSCZPLEX:MVS:SY	ITM_ManagedSystem	MS_Offline([Status="N" AND Reason<>"FA"] ON WSCZPLEX:MY	05/01/2009 11:02:42 PM	1	ITM Probl
WSCZPLEX:SYSB:M	ITM_ManagedSystem	MS_Offline([Status="N" AND Reason<>"FA"] ON WSCZPLEX:SYS	05/01/2009 11:02:42 PM	1	ITM Probl
hasl112	TEST_EVENT	Test message from hasl112	02/12/2009 02:18:19 PM	2	Problem
hasle332	e@09522621@09522621	Attempt to login as tbsmuser from host hasle332 failed	04/30/2009 02:35:39 PM	1	Problem
mwbt61	Administrator	Attempt to login as root from host mwbt61 failed	02/06/2009 06:19:51 PM	1	Problem
hasl112	TEST_EVENT	Test message from hasl112	02/12/2009 02:15:45 PM	3	Problem
hasl112	MWBTEST	Test Message	02/05/2009 05:36:58 PM	2	Problem
hasle332	Unix Event List	A e@09522621@09522621:1.0 process e@09522621@09522621:1.	05/05/2009 12:37:12 PM	1	Problem
hasle332	JJELD	A JJELD process running on hasle332 has connected as usemam	05/02/2009 05:17:33 PM	1	Problem
	RAD:Impact	A RAD:Impact process running on has connected as username r	05/02/2009 05:17:03 PM	1	Problem
	RAD:Impact	A RAD:Impact process running on has connected as username r	05/02/2009 05:16:59 PM	1	Problem
OPMGRC1	WARN_EVENT	fatal_error_on_guest	04/24/2009 11:26:56 AM	2	Problem
	Unix Event List	A e@OmnibusEventConnector process running on has connected	04/22/2009 11:51:58 AM	1	Problem
hasle332	Unix Event List	A e@09522621@09522621:1.0 process e@09522621@09522621:1.	04/20/2009 05:22:00 AM	1	Problem
hasle332	JJELD	A JJELD process running on hasle332 has connected as usemam	04/19/2009 11:39:32 PM	1	Problem
East	ATS_A_SrvGroup	Server1 experiencing problems	02/20/2009 07:23:37 PM	3	Problem
hasl112	TEST_EVENT	Test message from hasl112	02/12/2009 02:19:52 PM	1	Problem
hasl125	TESTEIF	test_message_from_eif_2	08/19/2008 03:30:51 PM	2	Problem
USIBMWZV.HSLV12	TBSMV3_SOURCE390		11/25/2008 05:23:22 PM	5	Problem
USIBMWZV.HSLV12	TBSMV3_SOURCE390		11/25/2008 05:23:21 PM	5	Problem

0 row(s) inserted, 1 row(s) updated and 0 row(s) deleted.

05/07/2009 08:13:57 PM root NCOMS[PRII]

Right clicking on the event and selecting “Information” will show the event fields. Here is a partial view of that information:

Event Information: Alert Status for Serial Number 314

Alert Fields | Alert Details | Journal

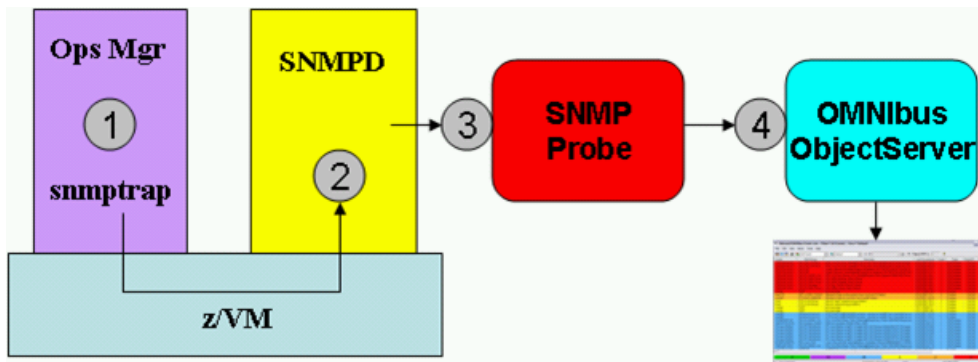
Agent: OpsMgr
 State Change: 05/05/2009 01:34:10 PM
 Alert Key: OpsMgr:postzmsg:tcp
 Local Sec. Obj.:
 TaskList: Not in Task List
 Local Root Obj.:
 Group: Public
 RAD_UserFunctionName: NCOMS314
 RAD_CurrentRawMetric:
 RAD_ServiceID: 0

guest_is_abending

Close Previous Next Help

SNMP Trap from z/VM to Netcool/OMNIBus

The general process flow for using the SNMP method is shown in the following diagram:



1. Based on a detected situation, Operations Manager creates an SNMP trap with the desired information and invokes the SNMPTRAP command provided in z/VM.
2. The SNMPD z/VM guest creates the trap and sends it to the SNMP Probe.
3. The SNMP Probe receives the event and applies its rules to map the event contents to the Netcool/OMNIBus alerts.status table.
4. Netcool/OMNIBus stores the event for use in its event management functions (display, correlate, trigger automation, etc).

z/VM SNMP Customization

Use of the SNMP method requires SNMP on z/VM to be customized (and preferably tested) before Operations Manager can use its functions. Details on z/VM TCP/IP SNMP customization are found in the *z/VM V5R4.0 TCP/IP Planning and Customization Guide* (SC24-6125)-05. The major steps that must be performed are:

1. The SNMPD service virtual machine, which is the SNMP agent for z/VM, must be active. This allows z/VM CMS guests to use the SNMPTRAP command.
2. The SNMP ports must be open. The following lines must exist in the PROFILE TCPIP member that is being used:

```
161  UDP  SNMPD                ;  SNMP AGENT
```
3. The trap destinations must be defined. This is done by creating a SNMPTRAP DEST file in the TCPIP 198 (D) disk, where each line of the file contains the hostname or IP of the trap receiver – in this case, it is the hostname/IP of the SNMP Probe (which is installed on the same platform as the Netcool/OMNIBus ObjectServer).
4. The MIB_DESC DATA and MIB_EXIT DATA files must reside on the TCPMAINT 198 (D) disk. MIB_DESC DATA contains the MIBs that will be active when SNMP is started. MIB_EXIT DATA contains your custom MIBs. Samples for both files can be copied from the TCPMAINT 591 disk.
5. The MIBX2DSC EXEC is required on the TCPMAINT 592 disk. This EXEC will merge your customizations from MIB_EXIT DATA into the operational MIB_DESC DATA file. The sample MIBX2DSC SAMPEXEC is shipped on TCPMAINT 592 disk, so it must be copied or renamed from SAMPEXEC to EXEC.

6. On TCPMAINT, for custom traps (which will likely be used), edit MIB_EXIT DATA to create mappings from a MIB variable name to a MIB variable number.
7. On TCPMAINT, run MIBX2DSC EXEC run to add your customizations from MIB_EXIT DATA into MIB_DESC DATA. This must be done whenever you change your customizations in MIB_EXIT DATA:

```
mibx2dsc MIB_EXIT DATA D MIB_DESC DATA D
```

Creating SNMP Traps

On z/VM, traps are created using the **SNMPTRAP** command, which is fully documented in the *z/VM TCP/IP User's Guide* (SC24-6333).

A trap is a formatted structure of variables (identified by an operator id (oid) a type (number, text, etc), and a value. A MIB defines and documents traps (including mapping the oids to a name) so that the function receiving the trap can understand its contents.

The types, format, and structure of traps are fully documented in the various RFCs that describe the SNMP versions (v1,v2,v2c, and v3) and SNMP MIBs; these are available from many sources on the internet which a simple search will uncover.

For this paper SNMPv1 traps are used. The key fields in SNMPv1 traps are:

- Enterprise—identifies the type of managed object that generates the trap.
- Agent address—provides the address of the managed object that generates the trap.
- Generic trap type—indicates one of a number of generic trap types. The standard generic types are coldStart, warmStart, linkDown, linkUp, authenticationFailure, and egpNeighborLoss, and Generic. Generic traps are most commonly used, as this allows anyone to develop custom traps for a monitored resource.
- Specific trap code—indicates one of a number of specific trap codes for the trap type.
- Time stamp—the amount of time that has elapsed between the last network reinitialization and generation of the trap.
- Variable bindings—the trap data field. Each variable binding associates a particular MIB object instance (oid) with its current value.

The z/VM SNMPTRAP command allows a simple method of creating a SNMPv1 trap, setting its type, associating data with the trap variables, and sending it to the trap destination. Here is an example of the SNMPTRAP command:

```
snmptrap trape 1.3 TICKS 12 1.3.7 text "Error 2pm" ent 1.6.7 inet 9.60.2.1 comm test
```


This sends an extended trap using an AF_INET (IPv4) socket to the SNMP Agent on host 9.60.2.1 that looks like this:

```
version      = SNMPv1
community    = test
enterprise    = 1.6.7
generic type  = 6 (enterpriseSpecific)
oid           = 1.3
specific type = 8 (TICKS)
value         = 12
oid           = 1.3.7
specific type = 9 (text)
value         = Error 2pm
```

Operations Manager Configuration

The Operations Manager service machines (OPSMGRSx) require

- IUCV authority defined in their directory entries. Add one of these statements to the directory entry of each OPMGRSx user ID:

```
IUCV SNMPD
IUCV ANY
```

- Access to the TCPIP 592 disk, which contains the SNMPTRAP command.
 - Add the following statement to the directory entry of each OPMGRSx user ID:
LINK TCPIP 592 592 RR
 - Add the following statement to the PROFILE EXEC of each OPMGRSx user ID:
'ACCESS 592 J'
(or choose a free file mode if J is already in use)

While SNMPTRAP can be called directly from the Operations Manager configuration file, it is best to have Operations Manager call a REXX EXEC that contains the SNMPTRAP command, to allow for additional event customization to take place. Here is a REXX EXEC called VM2LNX that receives the information from Operations Manager and sends a trap with specific characteristics:

```
/* REXX - send a snmp trap */
Trace O
Address Command
Parse Arg ":" msgtext
msgtext2 = '''msgtext '''
/* Send message */
snmptrap trape 1.1 number 30 1.2 text "UXZVM001" 1.3 text msgtext2 ent 1.3.6.1.4.1.9545.6
Exit
```

The SNMPTRAP parameters are on a single line. In the above example it is sending an extended enterprise specific trap 1.3.6.1.4.1.9545.6 and setting oid 1.1 to 30, oid 1.2 to "UXZVM001", and oid 1.3 to the contents of the msgtext variable.

As with the EIF method, the Operations Manager configuration file statements are customized to invoke the SNMPTRAP command, or a REXX EXEC containing the command, to create a trap:

```
* Send an alert to OMNIBUS using SNMP for abend msgs on consoles
DEFRULE NAME ABNDSNMP +
    MATCH '*abend*snmp*' +
    ACTION SNMPALRT
*
DEFACTN NAME SNMPALRT +
    COMMAND 'EXEC VM2LNX &T' +
    ENV SVM
*
```

In the above example, the Operations Manager configuration file statements will invoke the VM2LNX REXX EXEC and send a trap to Netcool/OMNIBus if the words “abend” and “snmp” are detected in the same line on any monitored console. Many other options are available, such as limiting which consoles are included in the rule, which text is included or excluded from the rule, etc.

- The DEFRULE statement defines the information to be looked for on the monitored consoles, and an action to be taken (SNMPALRT) when that information appears.
- The DEFACTN statement defines the SNMPALRT action, which invokes the VM2LNX EXEC. The &T parameter associated with the command EXEC VM2LNX is an Operations Manager predefined substitution variable for the DEFACTN command and substitutes the text of the message that triggered the action.

SNMP Probe

Details on the installation and configuration of the SNMP Probe are documented in the *IBM Tivoli Netcool/OMNIBus SNMP Probe* manual (SC23-6003).

The probe is installed on the same system with the Netcool/OMNIBus ObjectServer. It runs as a process on UNIX/Linux and can run as a service on Windows.

Although for this whitepaper SNMPv1 traps are used in the examples, the probe supports SNMPv1, SNMPv2/2c, and SNMPv3 traps.

After installing the SNMP Probe there are two files that must be configured:

- **mttrapd.props:** This is the properties file for the SNMP Probe. Information such as the listening protocol and port (normally the default of UDP 162), the location of the MIBs used for incoming traps, the location of the rules file, and the target Netcool/OMNIBus server are defined in this file.
- **mttrapd.rules:** This contains the rules for how the traps received by the SNMP Probe will be processed and mapped to Netcool/OMNIBus attributes before the event is sent to the Netcool/OMNIBus ObjectServer. The rules file syntax is documented in the *IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide* (SC23-6373).

The SNMP Probe builds a default set of variables from the trap that are used in its rules file. Some of the key variables for use in this specific process are:

\$enterprise	: V1 The SNMP enterprise string.
\$generic-trap	: V1 The SNMP generic trap integer value.
\$SNMP_Version	: V1, V2c, and V3 Has the value 1 for SNMP V1 traps and the value 2 for SNMP V2c traps.
\$specific-trap	: V1 The SNMP specific trap integer value.
\$Node	: hostname or IP address where the trap came from.
\$n	: the nth value in the varbind list
\$OIDn	: the nth oid in the varbind list
\$n_raw	: raw string representation of the nth varbind variable
\$n_text	: text representation of the nth varbind variable (with non-printable characters replaced with periods)
\$n_hex	: hex representation of the nth varbind variable

The contents of these variables can be examined by the rules file, and actions and/or OMNIBus event field mapping performed in the rules based on their contents.

Netcool/OMNIBus Configuration

The Netcool/OMNIBus server alerts.status table contains the Netcool/OMNIBus events. The fields in this table are the target of the SNMP Probe trap contents mapping. The default mttrapd.rules file performs the following mapping based on the trap type:

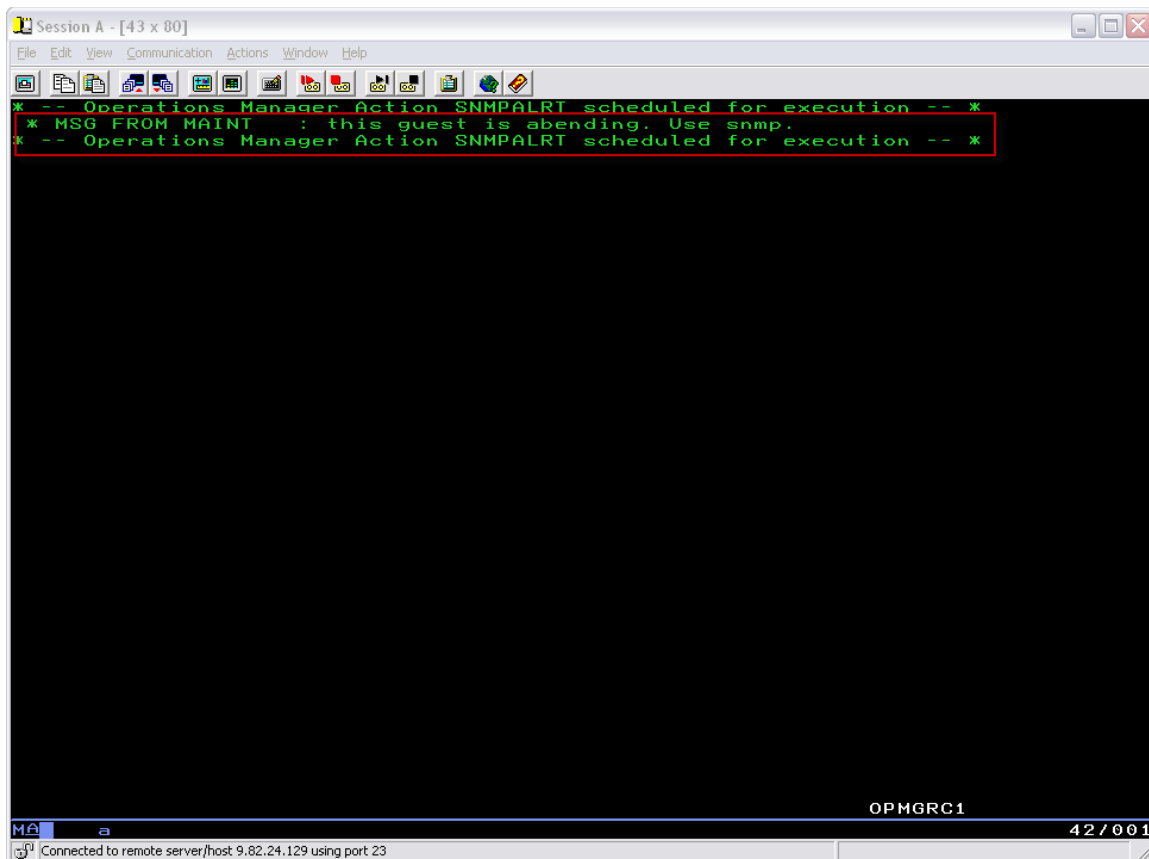
Trap type	AlertGroup	Summary	Severity	Identifier	Alertkey
0 (Cold Start)	Generic	Cold Start	4 (Major)	\$Node + “mttrapd” + \$generic- trap + \$specific- trap	
1 (Warm Start)	Generic	Warm Start	4 (Major)	\$Node + “mttrapd” + \$generic- trap + \$specific- trap	
2 (Link Down)	Generic	Link Down	5 (Critical)	\$Node + “mttrapd” + \$generic- trap + \$specific- trap + \$1	\$1 (first varbind variable value)
3 (Link Up)	Generic	Link Up	2 (Warning)	\$Node + “mttrapd” + \$generic- trap + \$specific- trap + \$1	\$1 (first varbind variable value)
4 (Authentication)	Generic	Authentication	3 (Minor)		
5 (Egp Neighbour Loss)	Generic	Egp Neighbour Loss	3 (Minor)		
6 (Generic)		Enterprise: + \$enterprise + Generic Trap: + \$generic- trap + Specific Trap: + \$specific- trap			

For trap type 6 – which is likely what most of the desired traps will be – custom rules will have to be added to place the desired information in the Alert Group, Severity, Identifier, AlertKey, and any other ObjectServer fields.

SNMP Trap Flow Example

This example will walk through how the components tie together to send a SNMP trap from z/VM to Netcool/OMNIbus.

1. Operations Manager detects an incident for which an event will be sent.
2. Operations Manager DEFACTN statement invokes a REXX EXEC that runs the SNMPTRAP command to create and send a SNMP trap:



```
Session A - [43 x 80]
File Edit View Communication Actions Window Help
* -- Operations Manager Action SNMPALRT scheduled for execution -- *
* MSG FROM MAINT : this guest is abending. Use snmp.
* -- Operations Manager Action SNMPALRT scheduled for execution -- *
OPMGR01
42/001
Connected to remote server/host 9.82.24.129 using port 23
```

3. The SNMP Probe processes the trap and sends it to the Netcool/OMNIbus ObjectServer. The assignment of the trap contents to the \$variables (they are listed without the "\$" prefix) can be seen:

```

Debug: D-UNK-000-000: [Event Processor] ReqId: 0
Debug: D-UNK-000-000: [Event Processor] enterprise: .1.3.6.1.4.1.9545.6
Debug: D-UNK-000-000: [Event Processor] generic-trap: 6
Debug: D-UNK-000-000: [Event Processor] specific-trap: 2
Debug: D-UNK-000-000: [Event Processor] UpTime: -2050538796
Debug: D-UNK-000-000: [Event Processor] Uptime: 259 days, 18:31:25.00
Debug: D-UNK-000-000: [Event Processor] community: GDP4
Debug: D-UNK-000-000: [Event Processor] IPaddress: 9.82.24.129
Debug: D-UNK-000-000: [Event Processor] PeerIPaddress: 9.82.24.129
Debug: D-UNK-000-000: [Event Processor] ReceivedPort: 162
Debug: D-UNK-000-000: [Event Processor] ReceivedTime: 1300286249
Debug: D-UNK-000-000: [Event Processor] Protocol: UDP
Debug: D-UNK-000-000: [Event Processor] SNMP_Version: 1
Debug: D-UNK-000-000: [Event Processor] OID1: .1.1
Debug: D-UNK-000-000: [Event Processor] 1: 30
Debug: D-UNK-000-000: [Event Processor] 1_raw: 30
Debug: D-UNK-000-000: [Event Processor] 1_text: 30
Debug: D-UNK-000-000: [Event Processor] 1_hex: 30
Debug: D-UNK-000-000: [Event Processor] .1.1: 30
Debug: D-UNK-000-000: [Event Processor] OID2: .1.2
Debug: D-UNK-000-000: [Event Processor] 2: UXZVM001
Debug: D-UNK-000-000: [Event Processor] 2_raw: UXZVM001
Debug: D-UNK-000-000: [Event Processor] 2_text: UXZVM001
Debug: D-UNK-000-000: [Event Processor] 2_hex: 55 58 5a 56 4d 30 30 31
Debug: D-UNK-000-000: [Event Processor] .1.2: UXZVM001
Debug: D-UNK-000-000: [Event Processor] OID3: .1.3
Debug: D-UNK-000-000: [Event Processor] 3: OPMGRC1: this is an abend for TDA please
alert via snmp
Debug: D-UNK-000-000: [Event Processor] 3_raw: OPMGRC1: this is an abend for TDA please
alert via snmp
Debug: D-UNK-000-000: [Event Processor] 3_text: OPMGRC1: this is an abend for TDA
please alert via snmp
Debug: D-UNK-000-000: [Event Processor] 3_hex: 4f 50 4d 47 52 43 31 3a 20 20 74 68 69 73
20 69 73 20 61 6e 20 61 62 65 6e 64 20 66 6f 72 20 54 44 41 20 70 6c 65 61 73 65 20 61 6c
65 72 74 20 76 69 61 20 73 6e 6d 70 20
Debug: D-UNK-000-000: [Event Processor] .1.3: OPMGRC1: this is an abend for TDA please
alert via snmp
Debug: D-UNK-000-000: [Event Processor] Node: 9.82.24.129
Debug: D-UNK-000-000: [Event Processor] PeerAddress: 9.82.24.129
Debug: D-UNK-000-000: [Event Processor] EventCount: 6
Debug: D-UNK-000-000: [Event Processor] Processing alert {0 remaining}
Debug: D-UNK-000-000: Flushing events to object server

```

4. Since this is a generic alert, the code below in bold was added to the mtrtrpd.rules file to produce a desired mapping to the Object Server. Based on the enterprise value, the AlertGroup and Summary fields are updated. The Severity is set to 5 if the third varbind variable is 30:

```

if (match($generic-trap, "6"))
{
    @Summary      = "Enterprise:" + $enterprise + "  Generic Trap:" + $generic-
trap + "  Specific Trap:" + $specific-trap + ""

    #code added for z/VM SNMP trap
    if (match($enterprise,".1.3.6.1.4.1.9545.6"))
    {
        @AlertGroup = "Z/VM_SNMP"
        @Summary = $3_raw + ": " + $enterprise + " " + $generic-trap + " " +
$specific-trap + ""

        update (@Summary)
        update (@AlertGroup)
    }
    if (match($1_raw,"30"))
    {
        @Type="1"
        @Severity="5"
        update (@Type)
        update (@Severity)
    }
}

```

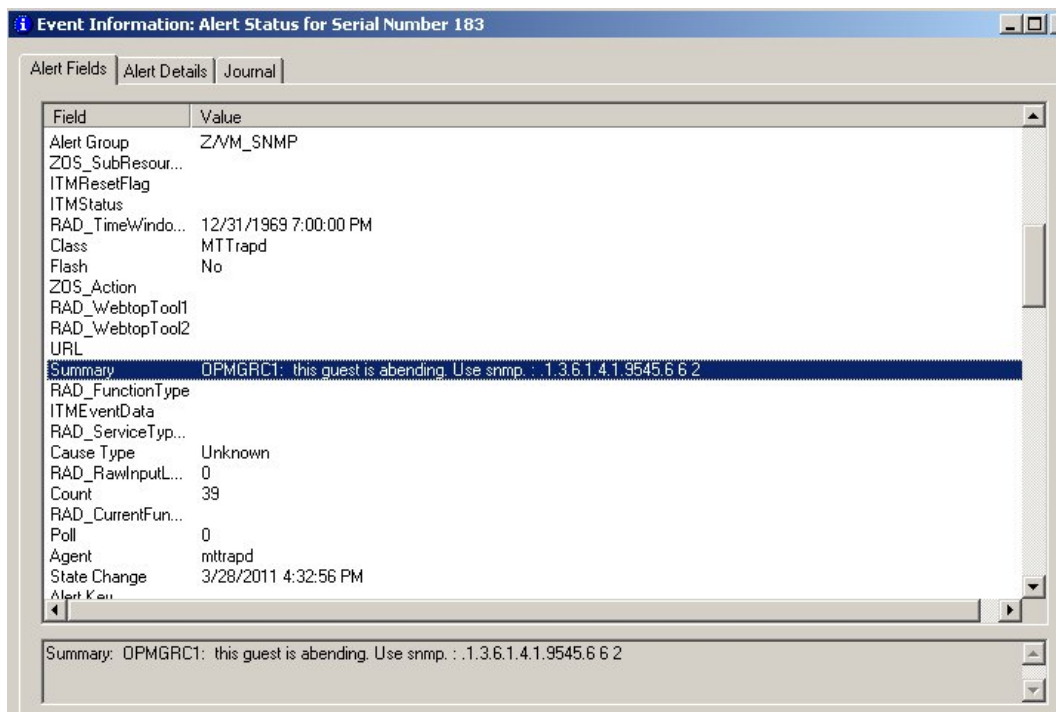
5. The event can now be seen on the ObjectServer console (and further event correlation/automation applied if desired):



The screenshot shows the Netcool/OMNIBus Event List window. The title bar reads "Netcool/OMNIBus Event List : Filter='All Events', View='Default'". The menu bar includes File, Edit, View, Alerts, Tools, and Help. Below the menu is a toolbar with icons for various actions. A search bar contains the text "Default". The main area displays a table with the following data:

Node	Alert Group	Summary
9.82.24.129	Z/VM_SNMP	OPMGRC1: This guest is abending. Use snmp: .1.3.6.1.4.1.9545.6.6.2

Right clicking on the event and selecting “Information” will show the event fields. Here is a partial view of that information:



Summary

The integration of IBM Operations Manager for z/VM and IBM Tivoli Netcool/OMNIBus enables the integration of z/VM and z/VM guest monitoring information into processes supported by enterprise wide event management, such as end-to-end availability and service monitoring.

- Use of the Event Integration Facility with Operations Manager allows integration with any event management environment that supports EIF events, so this can also be used to send events to event managers such as NetView on z/OS or Tivoli Enterprise Console. It also allows different triggering events to be sent to different targets.
- Use of SNMP with Operations Manager allows integration with any event management environment that supports SNMP trap management. All triggered events will be sent to the same target system.

These two capabilities allow Operations Manager monitoring to be leveraged well beyond the z/VM environment.

For Further Information

- IBM Tivoli Netcool/OMNIBus Event Integration Facility Reference (SC23-9686)
http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIBus.doc_7.3.0/omn_pdf_eif_master_73.pdf
- IBM Tivoli Netcool/OMNIBus Integration Best Practices
<http://www.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10NC10>
- Tivoli and Netcool Event Integration Flow
<http://www.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10EC01>
- IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide (SC23-6373)
<http://www.ibm.com/support/docview.wss?uid=pub1sc23637300>
- IBM Operations Manager for z/VM Administration Guide (SC18-9347)
<https://www.ibm.com/us-en/marketplace/operations-manager-for-zvm/resources>
- IBM Tivoli Netcool/OMNIBus SNMP Probe (SC23-6003)
http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIBus.doc/snmp-pdf.pdf