Tivoli Netcool Support's Guide to Rules file development

by

Jim Hutchinson

Document release: 2.1

Table of Contents

1 Introduction

This document covers the standard available rules files and the available rules file logic from the perspective of rules file development.

Rules file development requires knowledge of the incoming data, and the available functionality of the Netcool/OMNIbus rules file engine.

This document is provided as a supplement to the main product documentation.

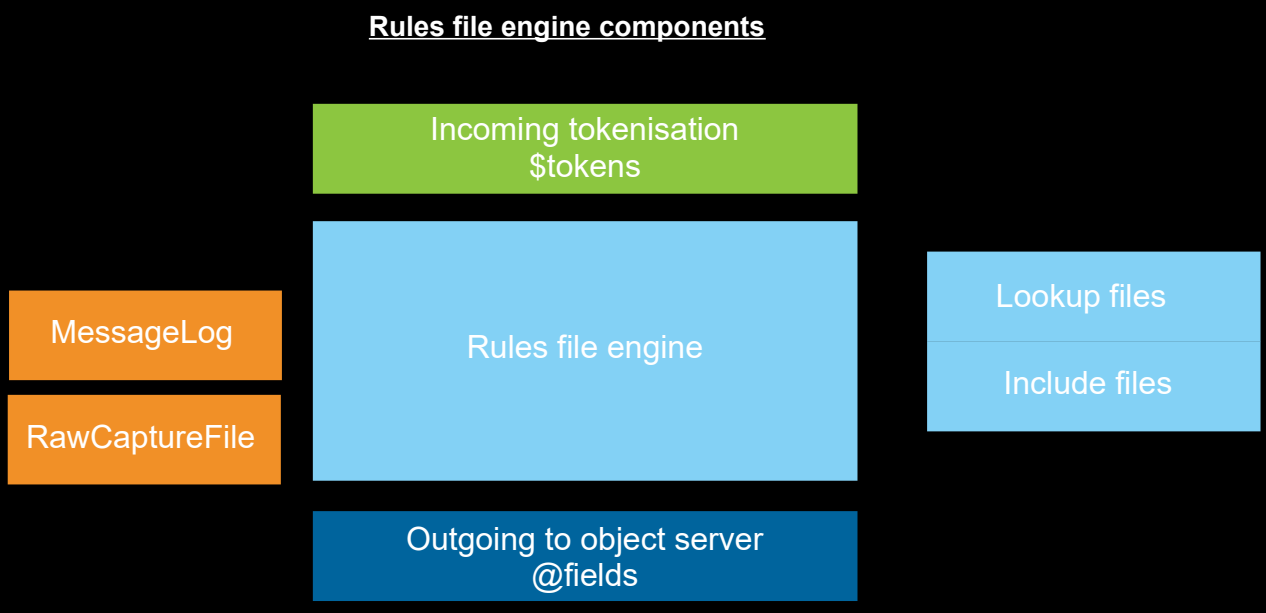
Useful documentation

- Tivoli Netcool/Omnibus 8.1 Probe and Gateway Guide [available features]
- Netcool/Omnibus 8.1 Administration Guide [alerts.status field definitions]

Refer to the Support's guide for rules file processing when common rules file logic examples are required.

1.1 Overview

The rules file engine is a bottle neck in the probes event processing, since the rules file engine is single threaded. It is important to ensure that the rules file logic uses exclusive logic, to prevent events exercising unrelated rules file logic.



1.2 $tokens – Input data

The $tokens hold the incoming data.
The probe specific tokens are defined in the probe manual in the 'element' section.
The RawCapture file includes the incoming $tokens and their values.

1.3 @fields – Output data

The @fields need to be defined in the target object server.
The @fields values are defined in the probes rules file logic to create unique events and hold the important information provided by the $tokens or the data looked up from lookup files.
The RawCapture file includes the @fields, with their initial values. They are marked with the pre-fix 'Status_'.

1.4 Syntax checking

The nco_p_syntax probe can be used to check most rules file logic.

Example usage.

```
$NCHOME/omnibus/probes/nco_p_syntax -server NCOMS -rulesfile ./myrulesfile.rules
-messagelevel info
```

The MTTrapd probe can use probe specific functions, such as SNMPGET|SNMPSET, which prevents the syntax probe from checking the rules file logic.

1.5 The RawCapture file

The RawCapture file is used to capture the $tokens and initial @field values.
The RawCapture property is set to 1 to enable raw file capture.

```
RawCapture                    : 1
RawCaptureFile                : '$NCHOME/omnibus/var/<name>.cap'
RawCaptureFileAppend          : 0
RawCaptureFileBackup          : 0
MaxRawFileSize                : -1
```

Example RawCapture file data for the Simnet probe.

```
Status_NodeAlias = ""
...
Status_Acknowledged = 0
Summary = "Link Up on port"
EventNumber = "1"
Group = "Link"
Agent = "LinkMon"
ServiceLevel = "2"
DateString = "DD/MM/YYYY HH:MM:SS"
Node = "link4"
Severity = "1"
RawCaptureTimeStamp = <UNIX timestamp>
```

Each event is separated by two newlines.

1.5.1 Replaying

The RawCapture file can be replayed to an object server using the Standard Input probe.

```
cat <probe>.cap | $NCHOME/omnibus/probes/nco_p_stdin -server NCOMS -rulesfile
./myrulesfile.rules -messagelevel info
```

The default RawCapture file includes the initial values of the @fields, which are not used within the RawCapture file replay. To reduce the file size and load, the @field 'Status_' rows can be removed using 'grep'.

```
grep -v Status_ <probe>.cap > <probe>.raw
```

Within the rules files being used in the replaying of the RawCapture data, the RawCaptureTimeStamp can be used to set the LastOccurrence, where event replication requires the historical timing.

```
cat <probe>.raw | $NCHOME/omnibus/probes/nco_p_stdin -server NCOMS -rulesfile
./myrulesfile.rules -messagelevel info
```

If full historical timing is required, the original RawCapture file needs to be used with the timing @fields set within the rules file logic.

For example.

```
if ( exists($Status_LastOccurrence) )
{
  @LastOccurrence = $Status_LastOccurrence
  @ProbeSubSecondId = $Status_ProbeSubSecondId
}
```

2 Rules file logic

2.1 Lookups

Using the lookup logic in the rules file.

```
@field = lookup($token,Example_t)
```

2.1.1 Lookup file

Lookup files can be defined within the rules file logic when the number of rows is small.

```
table Example_t=
{
 { "key1", "value1" },
 { "key2", "value2" },
 { "key3", "value3" }
}
default = {"default"}
```

2.1.2 Separate lookup file

For a larger number of lookup rows, the key field and values can be added to a separate file.

```
table Example_t = "$NCHOME/omnibus/probes/linux2x86/example.lookup"
default = {"default"}
```

With each key fields value being separated by the [TAB] character [\t].

File : $NCIHOME/omnibus/probes/linux2x86/example.lookup

```
key1[TAB]value1
key2[TAB]value2
key3[TAB]value3
...
keyn[TAB]valuen
```

2.2 Include rules files

The rules file logic can be held in multiple files by using the 'include' command.
It is best to use an environment variable, such as NCHOME, to define include rules file logic.

```
include "$NCHOME/omnibusprobes/include/probewatch.rules"
```

Within include rules files, the start and end of rules file logic can be marked with a log command.

```
log(DEBUG, "<<<<< Entering... <rulesfilename>.rules >>>>>")
…
log(DEBUG, "<<<<< Leaving... <rulesfilename>.rules >>>>>")
```

This makes debugging problems with the rules file logic easier, and is used in the NcKL rules files.

2.3 Functions

Refer to the Probe and Gateway Guide manual for the latest functions and general syntax.
This section will deal with the commonly used functions and their general usage.

2.3.1 Logging

The log function allows custom log messages to be logged to the probes log file. This is useful for tracking the event processing and debugging events.

```
Command : log([ DEBUG | INFO | WARNING | ERROR ],"string")
```

Example usage.

Logout the $tokens at message level informational.

```
foreach ( e in $* )
{
  log(info,"Debug: " + e + " = " + $e)
}
```

2.3.2 Maths functions

The maths functions allow the casting of $tokens and @fields when performing calculations.

```
if ( int($PercentFull) > 80 ) {
  $DiskSpace = ( real($diskspace)/ real($total) )*100.0
}
```

A good example of using the maths functions is the load rules file logic.

```
if ( match(%event_counter,"") )
{
        %start_time=getdate
        %event_counter = 1
        %total_event_counter = 1
}
else
{
        %end_time=getdate
        $time_elapsed = int(%end_time) - int(%start_time)
        %total_event_counter = int(%total_event_counter) + 1

        if (int($time_elapsed) > 10 )
        {
                $calculated_load = real(%event_counter) / real(int(%end_time) - int(%start_time))
                log(INFO,"EPS: " + $calculated_load + " (Total=" + %total_event_counter + ")" )
                %event_counter = 1
                %start_time=getdate
        }
        else
        {
                %event_counter = int(%event_counter) + 1
        }
}
```

2.3.3 Date/Time functions

The date/time functions allow the current UNIX time to be determined, along with converting date/time text to UNIX time, and vice versa.

```
$now      = getdate
$unixtime = datetotime("Tue Dec 19 18:33:11 GMT+00:00 2025", "EEE MMM dd HH:mm:ss ZZZZ yyyy")
$datetime = timetodate($unixtime, 'EEE MMM dd HH:mm:ss yyyy')

log(info,"DEBUG: now      = " + $now )
log(info,"DEBUG: unixtime = " + $unixtime )
log(info,"DEBUG: datetime = " + $datetime )
```

Example results:

```
DEBUG: now      = 1766169191
DEBUG: unixtime = 1766169191
DEBUG: datetime = Fri Dec 19 13:33:11 2025
```

2.3.4 The discard and recover functions

The discard function is used to prevent events being sent to the object server. With the recover function being used to undo any discard's performed earlier in the rules file logic. The rules file logic is worked through regardless of any discard function calls. It is therefore important to check if events were discarded before continuing with event processing. To reduce loading, performing discards early on in rules file logic is recommended.

The discarded function can be used to determine if the event has been discarded.

Example usage of the **discard, recover and discarded** functions:

```
If ( match($node,”localhost”) ) {
  discard
}

if( !discarded )
{
  log( DEBUG, "Continue processing..."
}

if(@Severity = 5 ) {
  recover
}
```

2.3.5 String functions

Rules file logic is used mostly for checking and processing strings, in order to present relevant data to the the end user.

Example usage of the **contains, match, nmatch, and regmatch** functions.

```
if ( contains( $Node, "host" ) ) {
  log(info,"Debug: contains = " + $Node)
}

if (match($Node, "omnihost")) {
  log(info,"Debug: matches = " + $Node)
}

if (nmatch($Node, "omni")) {
  log(info,"Debug: nmatches = " + $Node)
}

if ( regmatch($Node,"^omni[A-z][A-z][A-z][A-z]") ) {
  log(info,"Debug: regmatches = " + $Node)
}
```

Example results:

```
Debug: contains = omnihost
Debug: matches = omnihost
Debug: nmatches = omnihost
Debug: regmatches = omnihost
```

When

```
$Node = "omnihost"
```

The extract function is useful for collecting useful data from strings. When using the extract function is it recommended to use the regmatch function to confirm the tokens string matches the expected format.

Example usage of the **extract** function.

```
if (regmatch($tags,".*, HostName:.*, HostName:.*" ) )
{
  $hostname=extract ($tags, ".*, HostName:.*, HostName:(.*)"  )
  $hostname2=extract($tags, ".*, HostName:(.*), HostName:.*"  )

  log(info,"hostname=>"+$hostname + "<")
  log(info,"hostname2=>"+$hostname2 + "<")

} else if (regmatch($tags,".*, HostName:.*" ) )
{
  $hostname=extract($tags, ".*, HostName:(.*)"  )
  log(info,"hostname=>"+$hostname + "<")
}
```

Example results:

```
hostname=>localhost.company.com<
hostname2=>omnihost.company.com<
```

When

```
$tags = "[Environment]HOST:MONITORING, CLI:NETCOOL, [Environment]HOST:ABC], HostName:omnihost.company.com,
HostName:localhost.company.com"
```

After extracting text data, the data may required cleaning up or making uniform, to match with other $tokens or @fields.

Example usage of the **ltrim, rtrim, lower and upper** functions.

```
$ltrimNode = ltrim($Node)
$rtrimNode = rtrim($Node)

$lowerNode = lower($Node)
$upperNode = upper($Node)
```

Example usage to create a lower case $Node string:

```
$ltrimNode = ltrim($Node)
$rtrimNode = rtrim($ltrimNode)
$lowerNode = lower($rtrimNode)
```

Results:

```
Debug: lowerNode = >hostname<
```

When

```
$Node = " HostName  "
```

The **scanformat** function is useful when variables are found in a fixed format text string.

Example usage of using the **scanformat** function:

```
[$metric, $size] = scanformat($result, "%s = %s percent")

log(info,"Debug: metric is " + $metric + ", size is " + $size )
```

Results:

```
Debug: metric is Diskspace, size is 34
```

Where

```
$result = "Diskspace = 34 percent"
```

The **substr** function is useful when the string data is of a fixed column format.

Example usage of using the **substr** function:

```
#$16_hex ="12345678901234567890123456789012"

$h1 = substr($16_hex,1,4)
$h2 = substr($16_hex,5,4)
$h3 = substr($16_hex,9,4)
$h4 = substr($16_hex,13,4)
$h5 = substr($16_hex,17,4)
$h6 = substr($16_hex,21,4)
$h7 = substr($16_hex,25,4)
$h8 = substr($16_hex,29,4)

$ipv6 = $h1 + ":" + $h2 + ":" +$h3 + ":" +$h4 + ":" +$h5 + ":" +$h6 + ":" +$h7 + ":" +$h8

log(info,"Debug: ipv6 = " + $ipv6 )
```

Results:
```
Debug: ipv6 = 0001:4600:0000:8014:0000:2000:0000:5ba1
```

When

```
$16_hex ="00014600000080140000200000005ba1"
```

The **split** function is useful when the string data is a character separated list of items.

Example usage of using the **split** function:

```
array SplitArrayOfValues

 if (exists($1_text))
 {

   $input = $1_text

   $input = regreplace( $input, "^\,", " ," )
   $input = regreplace( $input, "\,$", ", " )
   $input = regreplace( $input, "\,\,", ", ," )

   $n = split( $input, SplitArrayOfValues, "," )

   foreach ( n in SplitArrayOfValues )
   {
     log(info,"Debug: SplitArrayOfValues = " + n + ':' + SplitArrayOfValues[n])
   }
 }
```

Results:
```
Debug: SplitArrayOfValues = 1:'Alarm has been generated.'
Debug: SplitArrayOfValues = 2:'10/02/2025 11:12:25'
Debug: SplitArrayOfValues = 3:'FAILURE'
Debug: SplitArrayOfValues = 4:'ERROR'
Debug: SplitArrayOfValues = 5:''
Debug: SplitArrayOfValues = 6:'999'
Debug: SplitArrayOfValues = 7:'1'
Debug: SplitArrayOfValues = 8:'omnihost'
Debug: SplitArrayOfValues = 9:'5'
Debug: SplitArrayOfValues = 10:'CRITICAL'
Debug: SplitArrayOfValues = 11:''
```

When
```
$1_text = "'Alarm has been generated.','10/02/2025
11:12:25','FAILURE','ERROR','','999','1','omnihost','5','CRITICAL',''"
```

The **regreplace** function is useful when there are unwanted characters in the string data.

Example usage of using the **regreplace** function:

```
# ALPHA NUMERICS [^A-Za-z0-9\\ ]
$EXPRESSION_RESULT = regreplace($TOKEN001 , "[^A-Za-z0-9\\ ]", "" )


log(info,"Debug: Alphanumerics only = >" + $EXPRESSION_RESULT + "<")
```

Results:
```
Debug: Alphanumerics only = >Actual data<
```

When
```
$TOKEN001="^L^P~@?Actual data^[^[^["
```

Using regreplace to decode an encoded URL string.

```
$result = regreplace($HTTP,   '%3A', ':')
$result = regreplace($result, '%2F', '/')
$result = regreplace($result, '%3D', '=')
$result = regreplace($result, '%3F', '?')
$result = regreplace($result, '%26', '\\&')

log(warn,"DEBUG : HTTP    = " + $HTTP)
log(warn,"DEBUG : result  = " + $result)
```

Input:
```
$HTTP    = http%3A%2F%2Flocalhost%2Fpath%3Fa%3Db%26c%3D1
```

Output:
```
DEBUG : result  = http://localhost/path?a=b&c=1
```

2.4 Name Value Pairs functions

The nvp_add function is used to create or update the name-value pair string of extended attributes.

Example usage of **nvp_add** to store the incoming $tokens in @ExtendedAttr:

```
@ExtendedAttr = nvp_add($*)

log(info,"Debug: @ExtendedAttr = " + @ExtendedAttr)
```

Results seen with the Simnet probe:

```
Debug: @ExtendedAttr = Agent="LinkMon";DateString="23/10/2025
11:08:43";EventNumber="1";Group="Link";Node="link4";ServiceLevel="2";Severity="1";Summary
="Link Up on port"
```

Example usage of **nvp_remove** to store the incoming $tokens in @ExtendedAttr:

```
@ExtendedAttr = nvp_add($*)

@ExtendedAttr = nvp_remove(@ExtendedAttr, "DateString")

log(info,"Debug: @ExtendedAttr = " + @ExtendedAttr)
```

Results seen with the Simnet probe:

```
Debug: @ExtendedAttr =
Agent="LinkMon";EventNumber="5";Group="Link";Node="link6";ServiceLevel="0";Severity="1";S
ummary="Link Up on port"
```

Example usage of **nvp_add** to store specific incoming $tokens in @ExtendedAttr:

```
foreach ( e in $Node, $Group, $Severity, $Summary, $Agent )
{
  @ExtendedAttr = nvp_add( @ExtendedAttr, e, $e )
}

log(info,"Debug: @ExtendedAttr = " + @ExtendedAttr)
```

Results seen with the Simnet probe:

```
Debug: @ExtendedAttr = Node="Sydney";Group="Systems";Severity="4";Summary="Machine has
gone offline";Agent="MachineMon"
```

2.4.1 Arrays

There is only one array function, the clear function.

```
clear(array_name)
```

Clearing arrays in rules file logic is usually performed on a 'kill -HUP'.

```
if( match( @Manager, "ProbeWatch" ) )
{
        switch(@Summary)
        {
        case "Running ...":
                @Severity = 1
                @AlertGroup = "probestat"
                @Type = 2
        case "Going Down ...":
                @Severity = 5
                @AlertGroup = "probestat"
                @Type = 1
        case "Rules file reread upon SIGHUP successful ...":
# Clear arrays on HUP signal
                clear(data_array)
        default:
                @Severity = 1
        }
        @AlertKey = @Agent
        @Summary = @Agent + " probe on " + @Node + ": " + @Summary
}
```

2.4.2 Host calls

Any external call made within the rules file can take a significant amount of time, and delay rules file processing. Because the rules file processing is single threaded, this makes host calls a performance risk, and should be avoided and reduced where possible.

There are three host call functions.

```
$My_Hostname = hostname()
@Summary = $Summary + " Node: " + $Node + " Address: " + gethostaddr($Node)
@Summary = $Summary + "Node: " + $Node + " Name: "+ gethostname($Node)
```

The most common usage is to find the probe servers hostname and IP address. This is can cause significant performance issues and can be avoided through the use of %static variables.

```
# Load the %static variables with the host call details
if(match(%MyHostname,""))
{
        %MyHostname = hostname()
        %MyIPHostname = gethostaddr(%MyHostname)
}

# Replaces host calls with the %static variables
@Manager = %Manager + "@" + %MyIPHostname
```

For gethostaddr and gethostname functions, arrays can be used to store the looked up values.

```
File : host_call_arrays.rules

array HostName
array HostIP

# If not set lookup values
if ( match(HostName[$Node],"") )
{
  HostName [ $Node ] = gethostname($Node)
  HostIP   [ $Node ] = gethostaddr($Node)
  log(debug,"Looked up details for $Node")
  log(debug,"HostName = " + HostName [ $Node ])
  log(debug,"HostIP   = " + HostIP [ $Node ])
}
```

2.4.3 Targets

The rules file logic allows events to be redirected to object servers, other than those defined in the probe property file. Additionally, more than one event can be created using the genevent function.

Available functions.

- registertarget
- setdefaulttarget
- settarget
- genevent

The extensions flood detection rules file logic uses the target feature and the genevent function.
The object server defined in the probe property file is defined as the target, DefaultOS.

```
DefaultOS = registertarget(%Server, "%ServerBackup", "alerts.status")
```

During a event flood the events are discarded.

The genevent function is used to forward performance related events to the object server, for use by the custom triggers.

The FloodEventsOS can be defined too.

```
FloodEventOS = registertarget("FLOODOS_V", "", "alerts.status")
```

Events can be redirected to the FlooodEventOS object server using the **settarget** function, under specific circumstances configurable by the flood.config.rules.

The genevent function

The genevent function needs to send events to object servers with the same target table fields, in type and order. If the object servers differ, specific fields can be specified to ensure the events are inserted correctly.

```
genevent(DefaultAlerts,
@Identifier, $MyIdentifier,
@Node , $MyNode,
@Summary, $MySummary,
@AlertGroup, $MyAlertGroup,
@AlertKey , $MyAlertKey,
@Type , @Type,
@Severity , @Severity,
@LastOccurrence, $now,
@ProbeSubSecondId, $MyProbeSubSecondId)
```

2.4.4 Arrays

Arrays can be used to hold dynamic data in memory. This is useful when not all the information is available in update events, or when earlier event states need to be checked.

Example usage for arrays is the discarding of events based on previously seen values. For example when events are seen due to a resynchronisation process.

File : resynchronisation.rules

```
############################################################
# DEFINE ARRAYS at the the top of the main rulesfile
############################################################
array   TimeStampLookup

###
# Determine if the event can be processed for discarding
$process_event_flag = 0

if(exists($NV_ALARM_ID)) {
  @Identifier = $NV_ALARM_ID
}
if(exists($NV_EVENT_TIME) && !match(@Identifier,"") )
{
  $process_event_flag = 1
}
############################################################
# Identifier MUST  be defined
# Compare $TimeStamp with stored value and discard the
# event if its the same as the stored timestamp
############################################################
if ( int($process_event_flag) == 1 )
{

# Set $TimeStamp to the UNIX timestamp from the event
  log(INFO,"NV_EVENT_TIME " + $NV_EVENT_TIME)
  $EVENT_TIME = datetotime($NV_EVENT_TIME, "MM/dd/yy hh:mm:ss")
  $TimeStamp = $EVENT_TIME

# Initialise array
  if ( match("",TimeStampLookup[@Identifier]) )
  {

    if (match(%ArrayCounter,"") )
    {
      %ArrayCounter = 1
    }
    else
    {
# Reset array if more than N elements
      if ( int(%ArrayCounter) > 100000 )
      {
        clear(TimeStampLookup)
        %ArrayCounter = 1
        log (INFO, "CLEARED TimeStampLookup")
      }
      else
      {
        %ArrayCounter = int(%ArrayCounter) + 1
      }
    }

  TimeStampLookup[@Identifier] = $TimeStamp
  log (INFO, "Stored NEW event = " + @Identifier)
  }
```

```
#########################################################
# Old event, then compare times
#########################################################
  else
  {
    $TimeStampLookup = TimeStampLookup[@Identifier]

# Process new event
    if ( int($TimeStamp) > int($TimeStampLookup) )
    {
      log (INFO, "Stroring NEW event time : TimeStampLookup = " + $TimeStampLookup + " : " + $TimeStamp )
      TimeStampLookup[@Identifier] = $TimeStamp
    }
# Discard old event
    else
    {
      log (INFO, "OLD event time : Discarding " + $TimeStamp + " <= " + $TimeStampLookup)
      discard
    }
  }
}
###
# Process events normally if not discarded
if( !discarded ) {

  log(DEBUG,"Processing event...")

}
```

2.5 Event definition

Events are defined by the @Identifier field in the object server. The choice of @fields or $tokens used for the depends on how the events need to be displayed to the end user [which includes reporting system].

There are two main methods for event handling.

- Generic clear
- Single event

2.5.1 Generic clear

The Generic Clear trigger uses the following fields by default.

Field	Comment
@Node	The events source node name e.g. The element managers server name
@Manager	The events source manager name e.g. The probes name
@AlertGroup	A string that defines the events grouping e.g. The source element manager
@AlertKey	A string that defines the event uniquely e.g. The alarm identification number
@Type	An integer that defines the events type e.g. 1 for problem, 2 for resolution

Example @Identifier setting.

@Identifier = @Node + ':' + @Manager + ':' + @AlertGroup + ':' + @AlertKey + ':' + @Type

The other fields considered by the Generic Clear trigger, not included in the @Identifier are.

- @Severity
- @LastOccurrence

The @Severity must be set to a value greater than zero, with the Resolution [@Type=2] event having a @Severity of indeterminate [@Severity=1] and the Problem [@Type=1] event having a @Severity of greater than clear [@Severity=0].

For a Problem [@Type=1] event to be cleared, it's Resolution [@Type=2] event must have a @LastOccurrence greater than Problem events @LastOccurrence.

The deduplication trigger discards any events that have older @LastOccurrence and @ProbeSubSecondId values that are not greater than the current event in the object server

- The expire trigger clears any events whose expire time has been reached
- The delete_clears trigger deletes any clear event that has a @StateChange value older than 2 minutes

In the multitier environment the standard triggers are replaced by layer specific triggers.
For example, the collection layer has these triggers.

- col_new_row
- col_deduplication
- col_expire

2.5.2 Single event

For single event handling, the @Type must not be set, and the problem and resolution events must have the same @Identifier.

Field	Comment
@Type	Must be unset, 0

The @Identifier must be set uniquely from the alarming object; this can be an alarm identifier, or a string of tokens that define the object.

An example single event usage is with the ITM events.

```
@Identifier = $situation_name + ":" + $situation_origin + ":" + $situation_displayitem + ":" + $ClassName
```

Typically the @Severity and @Summary provide the current state of the object, along with @LastOccurrence.

- @Severity The current severity of the object [0-5]
- @Summary A summary description of the objects state
- @LastOccurrence Either the probes time or the time of the event [must be set consistently]

3 Probe specific rules files

3.1 simnet.rules

The default simnet probe rules file logic uses a mix of Generic Clear for the Link Up|Down and single events for the other event types. Because the probe only creates single events, the use of ExpireTime makes sense, for clearing these events, in case the resolution event never arrives.

Example host problem/resolution event details.

```
@Identifier=hostlon01MachineMon2Systems
@Summary=Machine has gone online|offline
@Severity=2|4
```

File : simnet.rules

```
if( match( @Manager, "ProbeWatch" ) )
{
        switch(@Summary)
        {
        case "Running ...":
                @Severity = 1
                @AlertGroup = "probestat"
                @Type = 2
        case "Going Down ...":
                @Severity = 5
                @AlertGroup = "probestat"
                @Type = 1
        default:
                @Severity = 1
        }
        @AlertKey = @Agent
        @Summary = @Agent + " probe on " + @Node + ": " + @Summary
}
else
{
        @Manager        = "Simnet Probe"
        @Class          = 3300
        @Node           = $Node
        @Agent          = $Agent
        @AlertGroup     = $Group
        @Summary        = $Summary
        @Severity       = $Severity
        @Identifier     = $Node + $Agent + $Severity + $Group

        if (nmatch($Summary, "Port failure"))
        {
                @AlertKey = $PortNumber
        }
        else if (nmatch($Summary, "Diskspace"))
        {
                @AlertKey = $PercentFull + "% full"
        }

        if(regmatch($Summary, ".*Down.*")){
                @Identifier = "Down" + @Identifier
                @Type = 1
        }
        if(regmatch($Summary, ".*Up.*")){
                @Type = 2
        }
}
```

3.2 ping.rules

The ping probe is a polling probe, and collects the status of the hosts defined in the ping.file periodically.
This means that the number of events created per Poll is equal to the number of ping hosts.
The ping probes events are defined by the $host and the $status, by default.

The rules file logic can be adjusted to a single event by altering the @Identifier to $host, removing the setting of @Type and setting @Severity to 0 [clear] where the $status clears any previous $status.

File : ping.rules

```
if( match( @Manager, "ProbeWatch" ) )
{
        switch(@Summary)
        {
        case "Running ...":
                @Severity = 1
                @AlertGroup = "probestat"
                @Type = 2
        case "Going Down ...":
                @Severity = 5
                @AlertGroup = "probestat"
                @Type = 1
        default:
                @Severity = 1
        }
        @AlertKey = @Agent
        @Summary = @Agent + " probe on " + @Node + ": " + @Summary
}
else
{
        @Identifier = $host + $status
        @Node = $host
        @NodeAlias = $alias
        @Manager  = %Manager
        @AlertGroup = "Ping"
        @Class = 100

        switch ($status)
        {
                case "unreachable" :
                        @Severity = "5"
                        @Summary = @Node + " is not reachable"
                        @Type = 1
                case "alive" :
                        @Severity = "3"
                        @Summary = @Node + " is now alive"
                        @Type = 2
                case "noaddress" :
                        @Severity = "2"
                        @Summary = @Node + " has no address"
                case "removed" :
                        @Severity = "5"
                        @Summary = @Node + " has been removed"
                case "slow" :
                        @Severity = "2"
                        @Summary = @Node + " has not responded within trip time"
                case "newhost" :
                        @Severity = "1"
                        @Summary = @Node + " is a new host"
                case "responded" :
                        @Severity = "0"
                        @Summary = @Node + " has responded"
                default :
                        @Summary = "Ping Probe Error details : " + $*
                        @Severity = "3"
        }
}
```

3.3 message_bus.rules

The message bus probe can be used for many different integrations, so the default message bus rules file logic is an example of managing events generically. Specific integrations include rules file logic suitable for the integrations transport and expected $token's.

The rules file logic leverages the $TransformerName to allow standard include files to be included within a switch statement.

Where a transformer is not use the rules file logic matches $token's to @fields, one to one.

File : message_bus.rules

```
if( match( @Manager, "ProbeWatch" ) )
{
        switch(@Summary)
        {
        case "Running ...":
                        @Severity = 1
                        @AlertGroup = "probestat"
                        @Type = 2
        case "Going Down ...":
                        @Severity = 5
                        @AlertGroup = "probestat"
                        @Type = 1
        case "Start resynchronization" | "Finish resynchronization" | "Resynch Completed" :
                        @Severity = 2
                        @AlertGroup = "probestat"
                        @Type = 13
        case "Connection to source lost":
                        @Severity = 5
                        @AlertGroup = "probestat"
                        @Type = 1
        default:
                        @Severity = 1
        }
        @AlertKey = @Agent
        @Summary = @Agent + " probe on " + @Node + ": " + @Summary
}
```

```
else {
        @Manager = %Manager + " probe running on " + hostname()
        @Node = $Node
        @NodeAlias = %Host + ":" + %Port
        @Class = 30505

        if (exists($TransformerName))
        {
          switch($TransformerName)
          {
            case "dummy case statement": ### This will prevent syntax errors
                        include "message_bus_netcool.rules"
                        include "message_bus_cbe.rules"
                        include "message_bus_wbe.rules"
                        include "message_bus_wef.rules"
            default:
                log(DEBUG, "<<<<< Rules are not supported for this format >>>>")
                @Summary = "Rules are not supported for this format - " + $TransformerName
          }
        } else {
            log(DEBUG, "<<<<< Entering... message_bus_netcool.rules >>>>>")
            @Manager = %Manager
            @Class = 89210
            @Identifier = $Identifier
            @Node = $Node
            @NodeAlias = $NodeAlias
            @Agent = $Agent
            @AlertGroup = $AlertGroup
            @AlertKey = $AlertKey
            @Severity = $Severity
            @Summary = $Summary
            @StateChange = $StateChange
            @FirstOccurrence = $FirstOccurrence
            @LastOccurrence = $LastOccurrence
            @InternalLast = $InternalLast
            @Poll = $Poll
            @Type = $Type
            @Tally = $Tally
            @Class = $Class
            @Grade = $Grade
            @Location = $Location
            @OwnerUID = $OwnerUID
            @OwnerGID = $OwnerGID
            @Acknowledged = $Acknowledged
            @Flash = $Flash
            @EventId = $EventId
            @ExpireTime = $ExpireTime
            @ProcessReq = $ProcessReq
            @SuppressEscl = $SuppressEscl
            @Customer = $Customer
            @Service = $Service
            @PhysicalSlot = $PhysicalSlot
            @PhysicalPort = $PhysicalPort
            @PhysicalCard = $PhysicalCard
            @TaskList = $TaskList
            @NmosSerial = $NmosSerial
            @NmosObjInst = $NmosObjInst
            @NmosCauseType = $NmosCauseType
            @LocalNodeAlias = $LocalNodeAlias
            @LocalPriObj = $LocalPriObj
            @LocalSecObj = $LocalSecObj
            @LocalRootObj = $LocalRootObj
            @RemoteNodeAlias = $RemoteNodeAlias
            @RemotePriObj = $RemotePriObj
            @RemoteSecObj = $RemoteSecObj
            @RemoteRootObj = $RemoteRootObj
            @X733EventType = $X733EventType
            @X733ProbableCause = $X733ProbableCause
            @X733SpecificProb = $X733SpecificProb
            @X733CorrNotif = $X733CorrNotif
            @URL = $URL
            @ExtendedAttr = $ExtendedAttr
            @ServerName = $ServerName
            @ServerSerial = $ServerSerial
            log(DEBUG, "<<<<< Leaving... message_bus_netcool.rules >>>>>")
        }
}
```

3.4 tivoli_eif.rules

```
array servers;
array server_info;
array server_detail;

if( match( @Manager, "ProbeWatch" ) )
{
        switch(@Summary)
        {
        case "Running ...":
                @Severity = 1
                @AlertGroup = "probestat"
                @Type = 2
        case "Going Down ...":
                @Severity = 5
                @AlertGroup = "probestat"
                @Type = 1
        default:
                @Severity = 1
        }
        @AlertKey = @Agent
        @Summary = @Agent + " probe on " + @Node + ": " + @Summary
}
else
{
        switch($source)
        {
                case "dummy case statement": ### This will prevent syntax errors in case no includes are added
below.

                        # Uncomment the following include line when using the TotalStorage Productivity
            # Center integration:

            # Uncomment the following include line when getting the rules from Probe Extensions package
            # Uncomment the following include line when using the TPC integration:
                        # include "$NC_PROBE_EXT/eif/IBM_TPC/tivoli_eif_tpc.rules"


            # Uncomment the following include line when getting the rules from Probe Extensions package
                        # Uncomment the following include line when using the TSM integration:
                        # include "$NC_PROBE_EXT/eif/IBM_TSM/tivoli_eif_tsm.rules"


                        # Uncomment the following include line when using the TADDM integration
                        # rules file found in .../extensions/taddm/ :

                        # include "tivoli_eif_taddm.rules"

        #case "Director_Server"
            # Uncomment the following include line when getting the rules from Probe Extensions package
            # include "$NC_PROBE_EXT/eif/IBM_ISD/IBM_Systems_Director_Events.rules"

                default: ### We handle input from ITM here.
                        # ------------------------------------------------------------------
                        # Extract all of the fields we might require from the ITM situation.
                        # The ObjectServer will still work if extra quotes are left around
                        # the incoming fields but many of the products that OMNIbus integrates
                        # with will fail so remove them.
                        # ------------------------------------------------------------------

                        foreach ( e in $* )
                        {
                                if(regmatch($e, "^'.*'$"))
                                {
                                        $e = extract($e, "^'(.*)'$")
                                        log(DEBUG,"Removing quotes from attribute: " + $e)
                                }
                        }

                        # Default values for some key ObjectServer fields
                        # These may be overridden by the include rules files below
                        @Manager = "tivoli_eif probe on " + hostname()
                        if( exists( $ClassName ) )
                        {
                                @AlertGroup = $ClassName
```

```
                        }
                        @Class = 6601
                        if( exists( $source ) )
                        {
                                @Agent = $source
                        }
                        @Type = 1
                        @Grade = 1
                        if( exists( $severity ) )
                        {
                                switch ( $severity )
                                {
                                        case "FATAL":
                                                @Severity = 5
                                        case "60":
                                                @Severity = 5
                                        case "CRITICAL":
                                                @Severity = 5
                                        case "50":
                                                @Severity = 5
                                        case "MINOR":
                                                @Severity = 3
                                        case "40":
                                                @Severity = 3
                                        case "WARNING":
                                                @Severity = 2
                                        case "30":
                                                @Severity = 2
                                        default:
                                                @Severity = 1
                                }
                        }

                        # Uncomment the following include line when receiving events from TEC.
                        # Please note if the itm_event.rules file in used it may also populate TEC fields.
                        # include "$NC_PROBE_EXT/eif/IBM_TEC/tivoli_eif_tec.rules"

                        # This is the generic ITM situation handling. This may get modified if
                        # later include files are used. Must be included if the predictive or
                        # virtualization include lines are used.

                        # include "itm_event.rules"

                        # This is the ITM vmware agent situation handling to set BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "kvm_tbsm.rules"

                        # Uncomment the following include line to use the ITM predictive analytics
                        # integration files found in .../extensions/itmpredictive/ :

                        # include "predictive_event.rules"

                        # Uncomment the following include line and part 1 above to use the ITM
                        # virtualization integration files found in .../extensions/itmvirtualization/ :

                        # include "tivoli_eif_virtualization_pt2.rules"

                        # Uncomment the following include line to use the BSM Identity rules for ITCAM for SOA
                        # provided with TBSM in %OMNHOME%\probes\win23\tbsm_extensions on Windows
                        # and in $OMNHOME/probes/tbsm_extensions for non-windows

                        # include "kd4_tbsm.rules"

                        # This is the ITCAM Agent for WebSphere MQ situation handling to set BSM_Identity for
                        #  use with TBSM. The itm_event.rules file must also be uncommented when
                        #  this file is used.

                        # include "kmq_tbsm.rules"

                        # This is the ITCAM for Microsoft Applications - Active Directory Agent situation
handling to set BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "k3z_tbsm.rules"


                        # This is the ITCAM for Microsoft Applications - Exchange Server Agent situation
handling to set BSM_Identity for
```

```
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "kex_tbsm.rules"


                        # This is the ITCAM for Microsoft Applications - Hyper-V Agent situation handling to set
BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "khv_tbsm.rules"


                        # This is the ITCAM for Microsoft Applications - SQL Server Agent situation handling to
set BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "koq_tbsm.rules"


                        # This is the ITCAM for Microsoft Applications - Cluser Server Agent situation handling
to set BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "kq5_tbsm.rules"


                        # This is the ITCAM for Microsoft Applications - IIS Server Agent situation handling to
set BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "kq7_tbsm.rules"


                        # This is the ITCAM for Microsoft Applications - HIS Agent situation handling to set
BSM_Identity for
                        # use with TBSM. The itm_event.rules file must also be uncommented when
                        # this file is used.

                        # include "kqh_tbsm.rules"


                        # Uncomment the following include line to use the z/OS Event Pump rules
                        # provided with the z/OS Event Pump

                        # include "zos_event.rules"

                        # Uncomment the following include line to use the z/OS Event Pump user defined rules
                        # provided with the z/OS Event Pump

                        # include "zos_event_user_defined.rules"

                        # Uncomment the following include line to use the TBSM Identify rules for the
                        # z/OS Event Pump provided with TBSM in %OMNHOME%\probes\win23\tbsm_extensions on
Windows
                        # and in $OMNHOME/probes/tbsm_extensions for non-windows

                        # include "zos_identity.rules"


        }
}
```

3.5 mttrapd.rules

```
DefaultOS = registertarget(%Server, "", "alerts.status")

array OplTrapFloodHosts
array OplTrapFloodHosts_tmp
array OplTrapFloodHostsNosDroppedTraps
array OplTrapFloodHostsNosDroppedTrapsTimestamp

array ip_blockade_status_token

array snmp_watch_whole
array snmp_watch_payload
array watch_data_payload

if ( (exists($snmpreq) && !match($snmpreq,"")) ||
     (exists($snmp_action) && !match($snmp_action,"")) )
{
        include "mttrapd.bidir.rules"
}
else if( match( @Manager, "ProbeWatch" ) )
{
        $use_default_summary=1
        switch(@Summary)
        {
        case "Running ...":
                @Severity = 1
                @AlertGroup = "probestat"
                @Type = 2
                @AlertKey = @Agent
        case "Going Down ...":
                @Severity = 5
                @AlertGroup = "probestat"
                @Type = 1
                @AlertKey = @Agent
        default:
                if (nmatch(@Summary, "IP_BLOCKADE_STATUS"))
                {
                    $num_elements=split(@Summary,ip_blockade_status_token,"#")

                    if ( int($num_elements) >= 3)
                    {
                        switch(ip_blockade_status_token[2])
                        {
                            case "BLOCKED":
                                @Severity = 5
                                @Type = 1
                                @Summary = ip_blockade_status_token[3] + " is blocked"
                            case "UNBLOCKED":
                                @Severity = 1
                                @Type = 2
                                @Summary = ip_blockade_status_token[3] + " is unblocked"
                            default:
                                log( ERROR, "Unknown IP_BLOCKADE_STATUS state [" + ip_blockade_status_token[2] +
"]" )
                        }
                        @AlertKey = ip_blockade_status_token[3]
                        @AlertGroup = "Trap IP Status"
                        @Identifier = @NodeAlias + " " + @AlertKey + " " + @AlertGroup + " " + @Type + " " +
@Agent + " " + @Manager
                    }
                }
                else
                {
                    if (nmatch(@Summary,"SNMP_WATCH"))
                    {
                        include "mttrapd.snmpwatch.rules"
                        $use_default_summary=0
                    }
                    else
                    {
                        @Severity = 1
                        @AlertKey = @Agent
                    }
                }
        }
        if (int($use_default_summary)==1)
```

```
        {
            @Summary = @Agent + " probe on " + @Node + ": " + @Summary
        }
}
else
{
    @Manager = %Manager
    @Agent = "mttrapd"
    @Class = "300"

    ##########
    # Check if an SNMPv2 trap and convert to SNMPv1 style tokens
    ##########
    if(exists($notify)) ### if $notify exists then this is an SNMPv2 trap
    {
        @Node = $PeerAddress
        @NodeAlias = $PeerIPaddress

        if(regmatch($notify, "^enterprises\..*"))
        {
            $MIBFileNotNull = 1
            $notify = ".1.3.6.1.4.1." + extract($notify, "^enterprises\.(.*)")
        }
        else if(regmatch($notify, "\.iso\." ))
        {
            $notify = ".1." + extract($notify, "\.iso\.(.*)")
        }

        if (nmatch($notify, ".1.3.6.1.6.3.1.1.5")) ### Trap is an SNMPv2 Generic Trap
        {
            $enterprise = extract($notify, "(.*)\.[0-9]+$")
            $specific-trap = "0"
            $generic-trap = int(extract($notify, ".*\.([0-9]+)$"))-1
        }
        else ### Trap is Enterprise Specific
        {
            if(match(extract($notify, "\.([0-9]+)\.[0-9]+$"), "0"))
            {
                $enterprise = extract($notify, "(.*)\.[0-9]+\.[0-9]+$")
            }
            else
            {
                $enterprise = extract($notify, "(.*)\.[0-9]+$")
            }
            $specific-trap = extract($notify, ".*\.([0-9]+)$")
            $generic-trap = "6"
        }
    }
    else ### This is an SNMPv1 Trap
    {
        @Node = $Node
        if(exists($IPAddress)) ### Trap is from NNM or NV Probe
        {
            $IPaddress = $IPAddress
        }
        @NodeAlias = $IPaddress
        if(regmatch($enterprise, "^iso\..*"))
        {
            $NoQuietOutput = 1
            $enterprise = ".1." + extract($enterprise, "iso\.(.*)")
        }
    }



    @Identifier = "" + @NodeAlias + "" + @Agent + "" + $generic-trap + "" + $specific-trap + ""
    @Manager    = %Manager
    @Severity   = "1"


    if (match($generic-trap, "6"))
    {
          @Summary    = "Enterprise:" + $enterprise + "  Generic Trap:" + $generic-trap + "  Specific Trap:" +
$specific-trap + ""
    }
    else if (match($generic-trap, "0"))
    {
        @Summary = "Cold Start"
        @AlertGroup = "Generic"
        @Severity = "4"
    }
```

```
    else if (match($generic-trap, "1"))
    {
        @Summary = "Warm Start"
        @AlertGroup = "Generic"
        @Severity = "4"
    }
    else if (match($generic-trap, "2"))
    {
        # the $1 varbind will be ifIndex
        @Summary = "Link Down"
        @AlertGroup = "Generic"
        @Severity = "5"
        @Identifier = "" + @NodeAlias + "" + @Agent + "" + $generic-trap + "" + $specific-trap + "" + $1 + ""
        @AlertKey   = "" + $1 + ""
    }
    else if (match($generic-trap, "3"))
    {
        # the $1 varbind will be ifIndex
        @Summary = "Link Up"
        @AlertGroup = "Generic"
        @Severity = "2"
        @Identifier = "" + @NodeAlias + "" + @Agent + "" + $generic-trap + "" + $specific-trap + "" + $1 + ""
        @AlertKey   = "" + $1 + ""
    }
    else if (match($generic-trap, "4"))
    {
         @Summary = "Authentication"
         @AlertGroup = "Generic"
         @Severity = "3"
    }
    else if (match($generic-trap, "5"))
    {
         @Summary = "Egp Neighbour Loss"
         @AlertGroup = "Generic"
         @Severity = "3"
    }
    else
    {
         @Summary = "Trap type not matched"
    }

    ##############################################################################################
    # To effectively use mttrapd_flood_control.rules, TrapStat property must be enabled (set to 1).
    ##############################################################################################
    if ( int(%TrapStat) == 1)
    {
        include "mttrapd_flood_control.rules"
    }

    if ( int(%NoNameResolution) == 0 )
    {
        update(@Node)
    }
}
```

4 NcKL rules files

The Netcool/OMNIbus Knowledge Library rules files provides rules files for a set of commonly used devices, along with a rules file logic framework which can easily be added too, using the rules files created by the MIB Manager tool.

The rules file logic requires the NC_RULES_HOME environment variable to be set for the probe user.

4.1 MTTrapd probe rules file logic

The main NcKL rules file for the MTTrapd probe is the snmptrap.rules, which can be customized. In general it is best to add additional logic to the NcKL created @fields rather than modify the NcKL rules or MIB Manager generated rules file logic. Add custom rules file logic to the end of the main rules file logic, if required.

Main rules file logic

```
snmptrap.rules
snmptrap.adv.include.rules
snmptrap.sev.lookup
snmptrap.user.include.rules
```

Include directories with the includes and lookups

```
include-common   - Common trap rules file logic
include-compat   - Advanced correlation and compatibility rules file logic
include-snmptrap - Device specific rules file logic
```

The target registration definition can be updated to the probe property settings.

```
DefaultOS = registertarget( %Server, %ServerBackup, "alerts.status")
```

The Advanced correlation and compatibility rules file logic can be commented out in the snmptrap.rules.

```
##########
# Enter "Advanced" and "User" includes.
##########

# include "$NC_RULES_HOME/snmptrap.adv.include.rules"
# include "$NC_RULES_HOME/snmptrap.user.include.rules"

##########
# End of "Advanced" and "User" includes.
##########
...

##########
# Enter "compatibility" includes below with the following syntax:
#
# include "<$NCHOME>/etc/rules/include-compat/<rulesfile>.include.compat.rules"
##########

# include "$NC_RULES_HOME/include-compat/omnibus36.include.compat.rules"
# include "$NC_RULES_HOME/include-compat/AdvCorr36.include.compat.rules"

##########
# End of "compatibility" includes.
##########
```

4.2 Syslog probe rules file logic

The main NcKL rules file for the syslog and syslogd probe is the syslog.rules, which can be customized. In general it is best to add additional logic to the NcKL created @fields rather than modify the NcKL rules file logic. Add custom rules file logic to the end of the main rules file logic, if required.

Main rules file logic

```
syslog.rules
syslog-SrcType.lookup
```

Include directories

```
include-syslog - command and device rules file logic
```

The Advanced correlation and compatibility rules file logic can be commented out in the syslog.rules.

```
##########
# The following include statement is required by Netcool's advanced correlation
# logic.
##########

# include "$NC_RULES_HOME/include-syslog/CorrScore.include.syslog.rules"
# include "$NC_RULES_HOME/include-syslog/PreClass.include.syslog.rules"

##########
# End of advanced correlation include files.
##########

##########
# Enter "compatibility" includes below with the following syntax:
#
# include "<$NCHOME>/etc/rules/include-compat/<rulesfile>.include.compat.rules"
##########

# include "$NC_RULES_HOME/include-compat/omnibus36.include.compat.rules"
# include "$NC_RULES_HOME/include-compat/AdvCorr36.include.compat.rules"

##########
# End of "compatibility" includes.
##########
```

4.3 File sharing NcKL rules

The NcKL rules can be shared between the master and slave probes. The NcKL configuration provides a good example of how to configure the probe property files when file sharing is required.

In general, it is best to use the local file system, where the rules files are located. With the slave probe being the option for hosting the files, in terms of stability. The choice of whether to host the files on the master or slave server depends on the observed performance of the systems where the probes are being deployed. Factors, such as the number of file reloads, network load, and available resources determine which system is best to act as the file server.

In this example the same host is used for the master and slave probes, and the master probe uses the HTTP file server. This simplifies the configuration without affecting the overall design. The slave probe makes a useful file server as it will generally have a lower load compared to the master probe server. The slave probe can be used to check the rules file logic before forcing the master probe server to reload the rules files.

Note that the rules file cache file has a 1Gb limit, and includes the lookup files.

4.3.1 Master probe property file

The master probe uses the rules files published on the slave probe. To ensure the rules files are available when the slave probe is down, enable the probes rules file cache. Created the $NCHOME/omnibus/var/rulescache directory before starting the probe.

```
###
# Define the rules file using the HTTP format
RulesFile : 'http://0.0.0.0:10048/snmptrap.rules'
###
# Rules file cache directory rulesfilecache must exist, writes file mttrapd_master
CacheRules : 1
CacheRulesFile : "$NCHOME/omnibus/var/rulesfilecache/mttrapd_master"
# EOF
```

For the rules file logic to work the NC_RULES_HOME environment variable needs to be set correctly in the probe users environment.

```
NC_RULES_HOME=http://0.0.0.0:10048/
export NC_RULES_HOME

echo NC_RULES_HOME=$NC_RULES_HOME
NC_RULES_HOME=http://0.0.0.0:10048/
```

4.3.2 Slave probe property file

The slave probe acts as the rules file server, and is configured to use the local file systems rules file.

The NC_RULES_HOME environment variable is set to the location of the NcKL rules files.

```
NC_RULES_HOME=/opt/NcKL4.8/rules
export NC_RULES_HOME

echo NC_RULES_HOME=$NC_RULES_HOME
NC_RULES_HOME=/opt/NcKL4.8/rules
```

The probe property file defines local file system locations, and enables the file sharing of the NcKL rules.

```
###
# NcKL rules file location
RulesFile               : ''/opt/NcKL4.8/rule/snmptrap.rules'
###
# Command port
NHttpd.EnableHTTP               : TRUE
NHttpd.ListeningHostname        : '0.0.0.0'
NHttpd.ListeningPort            : 10048
NHttpd.AccessLog                : "$NCHOME/omnibus/log/mttrapd.10048.nhttpd.access.log"
###
# File serving NcKL rules
NHttpd.EnableFileServing        : TRUE
NHttpd.DocumentRoot             : '/opt/NcKL4.8/rules'
###
# Rules file cache directory rulesfilecache must exist, writes file mttrapd_slave
CacheRules : 1
CacheRulesFile : "$NCHOME/omnibus/var/rulesfilecache/mttrapd_slave"
# EOF
```

5 Extensions rules files

There are two main rules file extensions.

- Event flood management
- Return On Investment performance logging and table

5.1 Eventflood rules files

The Event Flood rules file logic can either discard events or else forward events to a FLOOD object server.

5.1.1 flood.config.rules

The flood.config.rules defines the target object servers and sets the flood.rules behaviour.

```
# Uncomment the FloodEventOS line and set the backup server name
# if you wish to divert events once an event flood has been detected.
DefaultOS = registertarget( %Server, %ServerBackup, "alerts.status")
#FloodEventOS = registertarget("FLOOD_VOS", "", "alerts.status")

array event_rate_array

# Calculate the average event rate over a number of seconds or
# max number of events received
$average_event_rate_time_window = 10
$average_event_rate_max_sample_size = 10000

# Calculate the current event rate for event flood detection
# over a number of seconds or max number of events received
$flood_detection_time_window = 5
$flood_detection_max_sample_size = 1000

# Do not begin flood detection until the probe has been running
# for a period.
$flood_detection_startup_time = 10

# Calculate the current event rate for anomalous event rate
# detection over a number of seconds or max number of events received
$anomaly_detection_time_window = 60
$anomaly_detection_max_sample_size = 10000

# Event rate threshold for flood detection in events per second
# Flood threshold takes probe from normal to flood condition
# Normal threshold should be lower to prevent flood condition oscilating
$flood_detection_event_rate_flood_threshold = 50
$flood_detection_event_rate_normal_threshold = 33

# Upper and lower event rate thresholds for determining anomalous
# event rate receipt
$lower_event_rate_threshold_multiplier = 0.10
$upper_event_rate_threshold_multiplier = 5

# Behaviour to take during an event flood
$discard_event_during_flood = 1

# If this element is set to 1, ensure that the registertarget statements
# at the beginning of this file are uncommented and contain a valid
# objectserver name.
$divert_event_during_flood = 0

# Forward events with severity of major or above to the primary
# objectserver during an event flood or divert events with a severity
# below major to a secondary objectserver.
$forward_event_minimum_severity = 4
```

5.1.2 flood.rules

The flood.rules performs the flood calculations.

5.1.3 flood_control.sql

```
File flood_control $NCHOME/omnibus/log/NCOMSFloodControl.log
```

table master.performance_stats

```
DatabaseName            varchar(64) primary key,
LastUpdate              UTC,
EventCount              int,
TriggerTimeCount        Real
```

table master.perf_per_minute

```
Minute                  int primary key,
EventRate               Real,
TriggerPercent          Real,
ClientPercent           Real
```

procedure set_probe_prop_http(in hostname char(255), in port int, in name char(255), in value char(255))
procedure set_probe_prop_https(in hostname char(255), in port int, in name char(255), in value char(255))
procedure set_prop_all_probes

trigger deduplicate_perf_per_minute
trigger objserv_stats_update
trigger flood_check

5.2 Return On Investment [ROI] rules files

The Return On Investment include rules file logic is used to log and store performance information.

5.2.1 probemanagement.sql

```
File: probemanagementlog $NCHOME/omnibus/log/NCOMS_probemanagement.log
```

trigger group probe_management

trigger probeevent_insert
trigger probeevent_reinsert

procedure do_probereloadrules (in address character(1), in port integer)
procedure do_probereloadrulesusingssl(in address character(1), in port integer)
procedure reloadrules_allprobes()

5.2.2 probestats.sql

```
File: probestats_report $NCHOME/omnibus/log/NCOMS_probestats.log
```

table master.probestats persistent

```
SeqNos                  incr,
KeyField                varchar(128) primary key, -- StatTime + ProbeId
StatTime                time,                     -- UTC Timestamp
ProbeUpTime             int,                      -- Probe Uptime in seconds
ProbeAgent              varchar(32),              -- Probe Agent field
ProbeHost               varchar(64),              -- Probe Host
ProbeId                 varchar(96),              -- Agent @ Host
ProbePID                int,    -- PID
NumEventsProcessed      int,    -- number of events entering rules file
NumEventsGenerated      int,    -- number of new events generated with genevent()
NumEventsDiscarded      int,    -- number of events discarded by rules file
RulesFileTimeSec        int,    -- in seconds
AvgRulesFileTime        int,    -- millionths of a second
CPUTimeSec              int,    -- in seconds
ProbeMemory             int     -- in kilobytes
```

table master.activity_probestats persistent

```
ReportAgent              char(32) primary key, -- objservlog or itmagent
ProbeStatsLastSeqNos     int,
MasterStatsLast          time
```

trigger probe_statistics_report
trigger probe_statistics_cleanup
trigger probestats_reset
trigger deduplicate_probestats

trigger probestat_insert
trigger probestat_reinsert

procedure probestats_data (in rowdata row of alerts.status)

5.2.3 probewatch.include

The probewatch.include file is designed to replace the logic in the ProbeWatch section of the main rules file.

```
if( match( @Manager, "ProbeWatch" ) )
{
# Replaces the Summary switch statement
  $NCHOME/omnibus/extensions/roi/probewatch.include
}
else
{
# Main rules file logic
}
```

Usually the probewatch.include file is copied to the rules file includes directory and the configurable actions modified as required.

```
# Configure actions for OplHeartbeat events
# None, one or many of these can be enabled

# Set if you want to discard the heartbeat events.
# By default discard Heartbeat events to maintain backwards compatability
$OplHeartbeat_discard=0


# Enable if you want to send probestat data to the master.probestats table
# stats_triggers trigger group must be enabled
$OplHeartbeat_populate_master_probestats=0

# Set if you want to log probestat data to the probe logfile at INFO level
$OplHeartbeat_write_to_probe_log=0

# Set if you want to generate any threshold events for probestat data
# Configuration of this is a manual step below
$OplHeartbeat_generate_threshold_events=0

# If set to 1, the Summary text will include HTTP port number as well as node.
# If set to 2, the Summary text will include HTTPS port number as well as node.
$prefix_with_http_port=0
```

5.3 Probe extensions rules files

The probe extensions rules files package provides a number of integrations for commonly used network management software, devices and equipment.

5.3.1 Generic event correlation

Generic event correlation is currently available for the following systems:
• Alcatel-Lucent 5620 SAM v13 [Nokia NFMP]
• Nokia-Siemens NMS2000
• Nokia-Siemens NetAct

Documentation description
Generic event correlation allows you to create event containers that correspond to incidents in networks and systems so you can determine the priority of the problems to be worked. The feature creates a one-to-one relationship between incidents and event containers so that first line operators can create trouble tickets from the head container events without duplication.

Generic event correlation consists of the following stages:

1. Pre-classification: This consists of assigning to each alarm a generic alarm type and defining its scope. It is achieved by adding the rules files supplied with the Probe Extension Package to the probe's rules file.

2. Containerization: This consists of assigning alarms to containers headed by a synthetic alarm. The containers retain the alarm's diagnostic information and enable easy access to the events grouped within the container. It is achieved by Netcool/OMNIbus automations.

3. Probable cause and impact analysis: This consists of setting the probable cause and impact of each alarm. For each alarm, the probable cause is determined by considering the highest weighted cause and impact from all the alarm's children events. It is achieved by Netcool/OMNIbus automations.

4. Presentation: This consists of displaying the event correlation information for a given alarm using the Web GUI Event Tables relationships feature.

```
correlation:
alcatel_5620_sam.genericcorr.include.rules
alcatel_5620_sam.genericcorr.user.include.rules
genericcorr.common.include.rules
netact.genericcorr.include.rules
netact.genericcorr.user.include.rules
netact3gpp.genericcorr.include.rules
netact3gpp.genericcorr.user.include.rules
```

5.3.2 Event enrichment

The event enrichment rules files are compatible with Probe for Alcatel-Lucent 5620 SAM V13 [Nokia NFMP] and Alcatel 5529 OAD. It was developed for the IBM Tivoli Network Manager 4.1.1 or higher.

```
eventenrichment/OAD5529:
alcatel_5529_oad_v6.enrichment.rules

eventenrichment/SAM5620:
alcatel_5620_sam.enrichment.rules
alcatel_5620_sam.itnm39.enrichment.rules
```

5.3.3 Tivoli EIF rules files

The Tivoli EIF rules files for the following integrations.

- IBM System Director events
- Tivoli enterprise console events
- IBM Tivoli StoragaProductivity Center events
- IBM Storage Manager events

```
eif/IBM_ISD:
IBM_Systems_Director_Events.rules
isd_db_update.sql

eif/IBM_TEC:
tec_db_update.sql
tivoli_eif_tec.rules

eif/IBM_TPC:
tivoli_eif_tpc.rules

eif/IBM_TSM:
tivoli_eif_tsm.rules
```

5.3.4 Generic Log File rules

ADVA FSP Network Manager is a network management and surveillance manager for ADVA's FSP series of optical and Ethernet products.

```
glf/ADVA_FSPNM:
adva-FSPNM.glf.lookup
adva-FSPNM.glf.rules
adva-FSPNM.glf_java.compat.rules
```

5.3.5 Internet of Things integration

Internet of Things using IBM Node-RED

```
iot/nodered:
flows
nodered.props
nodered.rules
nodes

iot/nodered/flows:
push2omnibus.json

iot/nodered/nodes:
omnibus

iot/nodered/nodes/omnibus:
EventFactory.html
EventFactory.js
EventFieldMap.html
EventFieldMap.js
```

5.3.6 socket rules

Configuration files for integration with Nokia NetAct 7 ASCII NBI Events.

```
socket/NSN_Netact:
NetactConfigParser.jar
netact.map.rules
netact.socket.lookup
netact.socket.rules
```

6 Appendix

6.1 Generic Log File rules for XML events

The example solution is for XML event data, where the event field data is presented on a new line.

Example event data.

```
<event>
        <nodealias>hostname123</nodealias>
        <summary>Test XML event</summary>
        <alertgroup>testing</alertgroup>
        <alertkey>TEST123</alertkey>
        <severity>CRITICAL</severity>
</event>
```

6.1.1 Probe property settings

```
Name                            : ' glf_java_xml'
RulesFile                       :
'$NCHOME/omnibus/probes/linux2x86/glf_java_xml/glf_java_xml.rules'
##
# Best practice
NetworkTimeout                  : 15
PollServer                      : 60
# Buffering - 20 events per second
Buffering                       : 1
BufferSize                      : 200
FlushBufferInterval             : 9
# Performance tuning
DisableDetails                  : 1
# Heartbeating
ProbeWatchHeartbeatInterval     : 60
##
# GLF Java probe property settings
LogFileName                     : '/tmp/xmllog.xml'
CleanStart                      : 'true'
ParserIgnoreEmptyFields         : 'false'
ParserElementDelimiter          : '<\/event>'
# ParserNVPDelimiter              : ''
# ParserQuoteCharacter            : ''
# ParserMaxEventSize              : 4098
# ParserStripCharacter            : ''
# ParserReplaceStripCharWith      : ''
RecoveryFile                    : '$NCHOME/omnibus/var/glf_java_xml.reco'
# EOF
```

6.1.2 Rules file logic

```
###
# Array to hold field values
array FieldName

###
# Define object server to send events to using genevent
DefaultOS = registertarget(%Server, %ServerBackup, "alerts.status","alerts.details")

###
# %static host lookups
if(match(%MyHostname,""))
{
        %MyHostname = hostname()
        %MyIPHostname = gethostaddr(%MyHostname)
}

###
# Standard GLF Java probe watch logic

if ( match( @Manager, "ProbeWatch" ) ) {
        switch(@Summary) {
                case "Running ...":
                        @Severity = 1
                        @AlertGroup = "probestat"
                        @Type = 2
                case "Going Down ...":
                        @Severity = 5
                        @AlertGroup = "probestat"
                        @Type = 1
                case "Log file truncated or overwritten: reopening":
                        @Severity = 2
                        @AlertGroup = "probestat"
                        @Type = 13
                        @ExpireTime = 300
                case "Log file disappeared: reading operation was aborted":
                        @Severity = 2
                        @AlertGroup = "probestat"
                        @Type = 1
                        @ExpireTime = 300
                case "Corrupted recovery file, proceeding with beginning of log file":
                        @Severity = 2
                        @AlertGroup = "probestat"
                        @Type = 1
                        @ExpireTime = 300
                case "Unable to read or write to recovery file":
                        @Severity = 4
                        @AlertGroup = "probestat"
                        @Type = 1

                default:
                        @Severity = 1
        }

        @AlertKey = @Agent
        @Summary = @Agent + " probe on " + @Node + ": " + @Summary
} else {

  @Manager    = %Manager
  @Class      = 7955
  @Node       = $Node
  @Agent      = "nco_p_glf_java"
  @Summary    = $*

  @NodeAlias  = $DeviceType

  @Identifier = @Manager + @AlertGroup + $Event

#  @FirstOccurrence = $Time
#  @LastOccurrence = $Time
```

```
###
# Check for XML events on the line
if (regmatch($FullDetails,".*<[A-z]+>.*<\/[A-z]+>.*"))
{
log (DEBUG,"DEBUG: XML line : " + $FullDetails)
###
# Extract field name and value
  $field_name = extract($FullDetails,".*<([A-z]+)>.*<\/[A-z]+>.*")
  $field_value = extract($FullDetails,".*<[A-z]+>(.*)<\/[A-z]+>.*")
###
# Store field details in array FieldName
  FieldName[$field_name] = $field_value

  log(DEBUG,"DEBUG: Field name : " + $field_name )
  log(DEBUG,"DEBUG: Field value : " + $field_value )

} else {

  if (regmatch($FullDetails,".*<event>.*"))
  {
    log(DEBUG,"DEBUG: *** <<< EVENT START" )
    clear(FieldName)

  } else if (regmatch($FullDetails,".*<\/event>.*")) {

    log(DEBUG,"DEBUG: *** <<< EVENT END" )

###
# Process event tokens and generate event
#
    log(DEBUG,"DEBUG: *** genevent processing ... " )

    foreach ( token in FieldName )
    {
      log(DEBUG,"DEBUG: Field Name : " + token )
      log(DEBUG,"DEBUG: Field Value : " + FieldName[token] )


      switch(token)
      {
       case "nodealias":
         $nodealias = FieldName[token]
       case "alertgroup":
         $alertgroup = FieldName[token]
       case "alertkey":
         $alertkey = FieldName[token]
       case "summary":
         @Summary = FieldName[token]
       case "severity":
           $severity = FieldName[token]
        default:
          log(DEBUG,"DEBUG: *** Field not used " + token + " = " + FieldName[token])
      }
    }
    @Node       = %MyHostname
    @Agent      = "XML processing"
    @Manager    = "GLF Probe"
    @AlertGroup = $alertgroup
    @AlertKey   = $alertkey
    @Identifier = @Manager + ":" + @Agent + ":" + @Node + ":" + @AlertGroup + ":" + @AlertKey + ":" + @Type

    @NodeAlias  = $nodealias

    switch($severity){
      case "CRITICAL":
        @Severity = 5
      case "WARNING":
        @Severity = 4
      case "INFO":
        @Severity = 2
      case "CLEAR":
        @Severity = 0
      default:
        @Severity = 2
    }
```

```
log(DEBUG,"DEBUG: @Identifier = " + @Identifier )

    genevent(DefaultOS,
    @Identifier, @Identifier,
    @Summary, @Summary,
    @Node, @Node,
    @NodeAlias, @NodeAlias,
    @Manager, @Manager,
    @AlertGroup, @AlertGroup,
    @AlertKey, @AlertKey,
    @Type, @Type,
    @Severity, @Severity )

    clear(FieldName)
  }

}
discard
}
```