NAS

Technical Brief

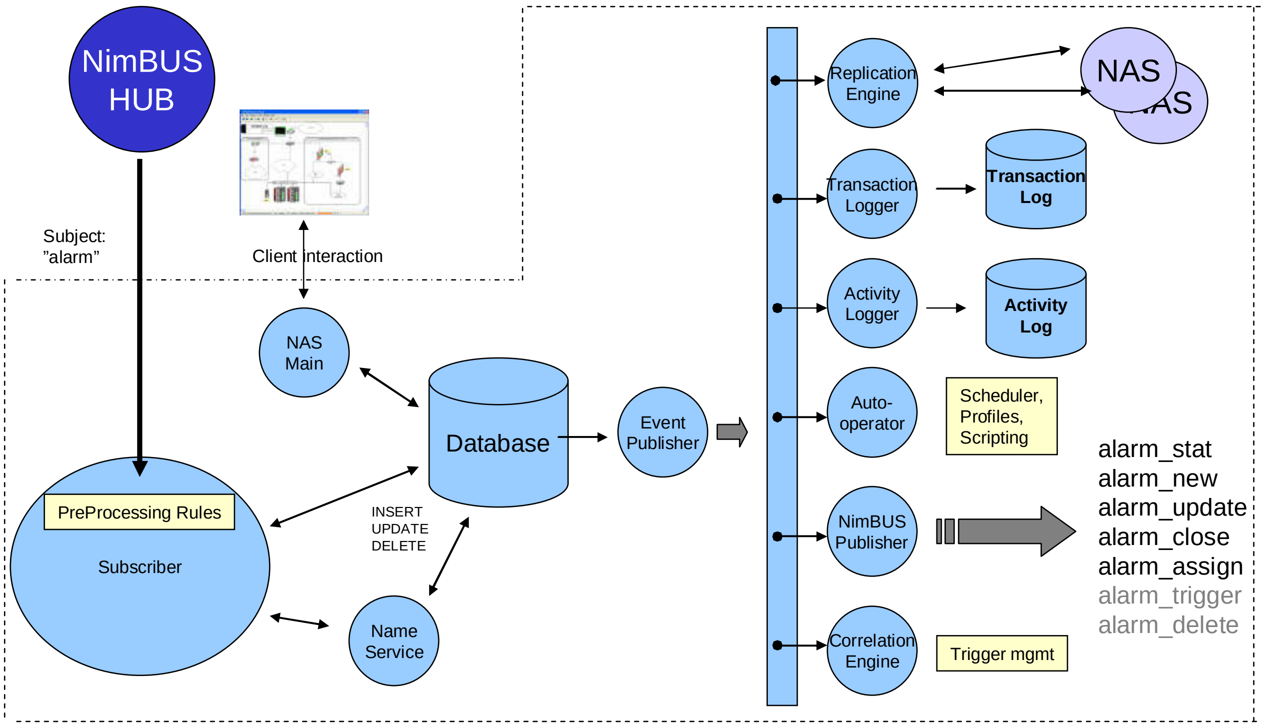
Version:	3.60
Date:	October 25, 2011
Author:	Carstein Seeberg

Table of contents

General

The NimBUS Alarm Server (NAS) subscribes to 'raw alarm' messages. These messages is generated by the NimBUS spooler under the 'alarm' subject. The NAS attaches to a hub queue named 'nas', and processes each incoming alarm according to its current configuration settings. The NAS performs pre-processing and post-processing of the alarms, through the auto-operator mechanism. The data is maintained in an embedded database and published back onto the NimBUS as the following events: *alarm_new, alarm_update, alarm_close, alarm_delete, alarm_assign* and *alarm_stats*. The target audience for these events are the NimBUS consoles and gateways.

Technical Overview



The NAS is built around the embedded database SQLite3 (www.sqlite.org) and an internal message pump. Each module runs in a separate thread receiving internal events. An incoming alarm message is received by the *subscriber* and checked against the pre-processing rules (custom, exlude or invisible). The ip-address in the alarm source field is looked up according to name resolution rules. The alarm message is then checked for possible suppression, and stored as a new or a updated record in the database. All changes to the database is transformed into events by database triggers and is stored in the NAS_EVENTS table. The events are read and placed on the internal message bus and removed from the table by the event publisher.

Subscriber

The subscriber connects the "nas" queue and decodes each incoming message. It will attempt to bulk-read as many messages as possible to increase speed. If suppression is enabled then the following algorithm is used to generate a MD5 suppression-key based on:

1) determine the *source* :
   alarm-source|message-source [**/**domain**/**robot]
2) determine suppression:
   a. *source***/**alarm-suppression-key or
   b. *source***/**alarm-message-text**/**alarm-sid**/**alarm-severity or
   c. alarm-message-text**/**alarm-sid**/**alarm-severity

The pre-processing rules are checked prior to any database operation. The "exclude" rule will prevent the event from any further processing. The "invisible" rule will turn the matching event into an invisible alarm, only when it is detected as new to the NAS. Finally the "custom" rule causes the NAS to run a pre-processing script allowing for advanced exclude/invisible possibilities as well as alarm-laundering for the fields *message, sid, source, hostname, user_tag1, user_tag2, visible* and *origin.* Please note that a positive match on any of the configured pre-processing rules inhibits further processing of rules for this event.

The subscriber will connect to the "nas" queue which typically is configured to subscribe for the "alarm" subject. As of version 3.25 it is possible to configure for a different subject than "alarm", this is meant for environments that is in need of a pre-processing/laundering filter in front of the NAS. You can also specify multiple subjects by separating them with a comma.

The following user-data fields in the incoming alarm are used: *level, subsys, source, message,* and *custom_1* to *custom_5*.

The subscriber (NAS 3.60) will pick *subscr_block_size* from the NAS queue and operate on this as a single transaction. The default value is 0 (disabled), causing a singular (i.e. non-bulk) message dispatching. Please note that the block size needs to by synchronized with the HUB's *setup/bulk_max_size (default: 1000).* So if you want to increase the block size to 3000 then you need to configure the NAS (*setup/subscr_block_size)* and the HUB (*setup/bulk_max_size).*

Failure to insert messages into the database, caused by malformed packets etc. will be stored into the *failrepo* directory. The fileformat is PDSFILE with the nimid as the filename. These files can be decoded with the various API's and the built-in script language, but is also somewhat readable in notepad.

Name Service

The hostname is determined by the following ip-to-hostname resolution schema below. Note that this behavior is enabled/disabled in the NAS UI Name Services properties.

NimSource	NimRobot	AlarmSource	Source	Hostname
Yes	Undefinded	Undefined	NimSource	Lookup
Yes	Undefined	Yes	AlarmSource	Lookup
Yes	Yes	Undefined	NimSource	NimRobot
Yes	Yes	Yes	AlarmSource	NimRobot or Lookup$^{1)}$

$^{1)}$ An IP to name resolution is performed when the AlarmSource differs from the NimSource and the Source is a valid IP address. Otherwise NimRobot is used.

Replication Engine

This module is responsible for replicating alarm data, script files and configuration data to its replicating endpoints by export and import mechanisms. The replication data is stored in a outgoing queue in the replication directory structure. Each queue is a SQLite database containing two tables, REPL_CONFIG and REPL_QUEUE. The current replication configuration is serialized and stored in the REPL_CONFIG table. Changes to the configuration will result in a full cold-start of the queue i.e the queue file is removed.

Events will be extracted from the internal bus and stored into the various queues along with files and configuration data. Alarms that have a complete transaction (new + close) in the queue will <u>not</u> be translated to console events. The queues will be bulk exported through the NimBUS request *repl_queue_post*. The bulk size is configurable, and is default 2000 queue items. The bulk data is stored on the remote NAS as replication/*.import file.

A separate ‘replication importer’ thread is started and will attempt to read the import file from all configured alarm servers in the replication schema on interval and will process the whole import file as quickly as possible.

Transaction Logger

This module is only started if the NAS transaction logging is activated. It converts internal events to database records stored in the transactionlog database (transactionlog.db). The following tables are maintained by the transaction logger:

NAS_TRANSACTION_LOG	Stores the actual alarm event as received and processed by the NAS. This is enabled by the ‘Log transaction details’ flag. Compression is done by removing suppressed updates. Houskeeping is done by removing the entire transaction.
NAS_TRANSACTION_SUMMARY	One record per unique alarm is maintained and updated. Housekeeping is done by removing the alarm record.

Transaction-log filtering
The transaction-logger now has the facility to filter out events based on the same criteria as the pre-processing filters exclude, custom and invisible. This will prevent repetitive events e.g. heartbeats to be stored in the transaction-log which will conserve space and reduce overhead.

Activity Logger

This module maintains the NAS_ACTIVITY_LOG table in the transactionlog.db database. It is configurable for the user through the activity logger setup dialog.

Correlation Engine

Builds and maintains the triggering data-structures used for correlation rules and the auto-operator mode *on_trigger*. The trigger data-structures are maintained in memory and are usable for the auto-operator and the scripting language.

Auto-operator

The auto-operator (AO) is configured through AO profiles and pre-processing filters. It also controls the built-in scheduler. The AO profiles are acted upon according to its type and limited through the operating-period (if any).

NimBUS Publisher

Generates the external events for NimBUS gateways and consoles. The following events are generated: alarm_new, alarm_update, alarm_close, alarm_stat, alarm_trigger and alarm_delete. Note that the *alarm_delete* message needs to be activated by the configuration parameter setup/pub_alarm_delete.

NiS Bridge

The NAS bridge retrieves the connection-string from the data_engine and maintains the NAS_ALARMS, NAS_NOTES, NAS_ALARM_NOTE, NAS_TRANSACTION_SUMMARY and NAS_TRANSACTION_LOG tables stored in the MS-SQL server. The NAS attempts to synchronize the embedded database and the NiS database during its startup procedure. The NAS will automatically make the bridge visible in the UI if the NiS is detected. The latest addition to the NiS bridge is the ability to monitor the computer system (CS) table (CM_COMPUTER_SYSTEM) for changes in the *state* field. This field indicates whether the computer system (or rather the device(s) that is referenced by the CS) is in a maintenance/managed mode or not. As part of the upgrade to make the probes compliant to the NMS 4.1 data-model, then the alarms will be tagged with the sender device_id. This is then mapped to the appropriate CS (over the CM_DEVICE table) and a state may be examined and used by the current computer system monitor ruleset. The main purpose of this functionality is to act towards alarms from the device regardless if it is monitored locally or from one or more remote locations from different probes.

Internationalization

The NAS 3.50 supports the internationalized alarms published by probes supporting the new NIS Configuration-Item bindings. The incoming alarm PDS is expanded with *i18n_token* and *i18n_data* elements. The NAS will, if *internationalization* is enabled, store the *i18n_token* and the base64 encoded PDS byte-stream *i18n_data* in the database. The *get_alarm* command-interface and the event-publisher will decode the base64 PDS into a valid PDS called *i18n_data*. The i18n tokens cannot be used as auto-operator filters, and is purely meant for the UI's.

Storm Protection

The NAS 3.60 supports a built in storm protection feature that will prevent large continuous event storms from a robot or probe to cause problems for the NAS. The algorithm is constructed in a way that the NAS maintains a "quarantine list" for possible offenders. The size of this list is configurable (**storm_capacity**) and elements will be added or removed or moved to the top depending on the message frequency. The event "signature" is constructed by e.g. *source, domain, robot [,probe-id [,supp_key]* elements of the inbound alarm message. If the number of alarms matching the "signature" exceeds a threshold (**storm_threshold**) within a specified time-window (**storm_timewindow**) then succeeding alarms will be quarantined by republishing the message to configured subject (**storm_subject**). The default subject is NAS_QUARANTINE. The quarantined alarm will **not** be registered with the NAS. An and a log-entry is generated when the first messages is placed in quarantine. The alarm message text and severity level can be overridden (**storm_message, storm_severity_level**)

The **storm_protection** value causes the key "signature" elements to be:

0. disabled
1. source, domain, robot, probe-id and supp_key
2. source, domain, robot, probe-id
3. source, domain, robot

The *storm_message* string supports <u>variable expansion</u> from the message header, e.g.

```
Placing alarm(s) from $domain:$origin:$robot:$prid:suppkey=$supp_key,  total:%d
```

Configuration and Data Files
The following configuration options are not available through the UI:

Section/Variable	Values	Default	Description
setup/public_get_alarms	yes|no	no	require NimBUS permissions to get_alarms
setup/script_dir	<path>	scripts	root-level directory for scripts
setup/pub_alarm_delete	yes|no	no	Publish alarm_delete events
setup/pub_alarm_invisible	yes|no	no	Publish invisible alarm events
setup/data_engine	<probe>	data_engine	
setup/distsrv	<probe>	distsrv	Address for distsrv.
setup/transactionlog_no_vacuum	yes|no	no	Skip translog housekeeping.
setup/transactionlog_housekeeping	<tspec>	00:30[1)]	RFC2445 - timespecification.
setup/subject	<string>	alarm	For pre-processing purposes.
setup/dbs_block_size	<integer>	1000	The number of items that will be unqueued from the internal database request queue per cycle.
setup/subscr_block_size	<integer>	3000	The number of alarms to read from the NAS queue in a single read.
setup/subscr_yield_at	<integer>	5000	Number of elements in internal event queue before subscribe should yield.
setup/subscr_yield_regain	<integer>	1000	Number of elements in internal event queue before subscriber should regain control.
setup/reset_suppcount_on_severity	yes|no	no	Resets the suppression counter to zero when the severity changes.
setup/use_full_sid_path	yes|no	no	Expands subsystem-id to full subsystem path.
setup/custom_headers/custom_1	string	Custom 1	Set header caption (1-5)
replication/num_failures	<integer>	20	Failures before sending alarm.
replication/queue_reconnect	<integer>	60	Reconnect freq. in seconds
replication/service_interval	<integer	60	Freq. for checking changes to scripts and configuration.
replication/service_monitor	<integer>	900	Repl. queue size monitor.

[1)]The transactionlog_housekeeping specifies when the database housekeeping (vacuum) is done. This string is base on the RFC2445 recurring schedule format, and is the same that is used by the NAS scheduler.

The default specification is every night at 00:30 (specified with the syntax):
rfc2445|RRULE:FREQ=DAILY;INTERVAL=1;BYHOUR=0;BYMINUTE=30

Please refer to the Appendix section for the data models used by the NAS.

Command Interface Description

This sections describes the NAS command interfaces. All commands will return a status value like NIME_OK (0), NIME_ERROR (1) or NIME_INVAL (7). Specialties are documented under each command. The 'list' type commands yields a PDS data-structure.

assign_alarms

Parameter	**Type**	**Req**	***Description***
by	string	*	*specifies who assigned the alarm(s)*
to	string	*	*specifies to whom the alarm is assigned to*
nimid	string	*	*alarm message-id*
nimids	*array*		*a string table of message-ids*

close_alarms

Parameter	**Type**	**Req**	***Description***
by	string	*	*specifies who closed the alarm(s)*
nimid	string	*	*alarm message-id*
nimids	*array*		*a string table of message-ids*

date_forecast

Parameter	**Type**	**Req**	***Description***
specification	string	*	*RFC-2445 compliant string*
startdate	string		*ISO starting date of forecast. yyyy-mm-dd hh:mm:ss*
nitems	number		*number of dates in forecast.*
format	*string*		*strftime format specifiers.*

This command returns a string array with dates, and the number of dates in the forecast. The current time is used as the default startdate.

db_query

Parameter	**Type**	**Req**	***Description***
sql	string	*	*SQL-92 conformant statement*
db	*string*	*	*database*

get_alarms

Parameter	**Type**	**Req**	***Description***
show_all	number		*flag showing all alarms visible and invisible*
origin	string		*filter alarms using origin field*
hostname	string		*filter alarms using hostname field*
source	string		*filter alarms using source field*
severity	string		*filter alarms using severity field*
subsystem	string		*filter alarms using subsystem field*
assigned_to	*string*		*filter alarms using assigned_to field.*

The show_all parameter takes the following values:
0: show only visible alarms.
1: show all alarms with visibility flag in alarm record.

All filter items are *italic* and may be used together. The following syntax is assumed:

[not] [like] value [,value [...]]] | null

e.g assigned_to = *not null*
assigned_to = administrator
assigned_to = *null*
hostname = *like* %xp%

Compatibility note:
The mask parameter (used by e.g the Alarm Notifier) is supported as a non-public variable, hence not being visible.

get_ao_status

Parameter	Type	Req	Description
mode	string		*a combination of triggers, profiles, schedules and filters.*
name	string		*filter the selection on name.*
detail	number		*detail = 1 will only list active selections.*

get_info

Parameter	Type	Req	Description
detail	number		*shows current connections if set to 1.*
show_all	number		*flag showing all alarms visible and invisible*
origin	string		*filter alarms using origin field*
hostname	string		*filter alarms using hostname field*
source	string		*filter alarms using source field*
severity	string		*filter alarms using severity field*
subsystem	string		*filter alarms using subsystem field*
assigned_to	*string*		*filter alarms using assigned_to field.*

Please see get_alarms for the parameter settings.

get_sid

Parameter	Type	Req	Description
sid	*string*		*specific subsystem identifier e.g. 1.1.1*

Returns all subsystem names configured or the one specified by *sid*.

host_summary

Parameter	Type	Req	Description
mode	*string*		*one of: today, lasthour, last24hours, last3days, lastmonth and date=ISO-startdate,ISO-enddate. E.g. date=2007-08-24,2007-08-27*

Returns a list of hosts that has alarms in the period specified by the *mode*.

nameservice_create

Parameter	Type	Req	Description
ip	string	*	*ip-address to be used as lookup key.*
name	string	*	*name to be used in name-resolution*

lock	*number*		*specifies if this should be locked (1=locked)*

Adds a nameservice record.

nameservice_delete

Parameter	Type	Req	*Description*
ip	*string*	*	*ip-address to be removed.*

nameservice_list

Returns a PDS table (named *table)* with records containing *ip,name,ts* and *time.*

nameservice_lookup

Parameter	Type	Req	*Description*
ip	string		*ip-address to resolve.*
name	*string*		*hostname to resolve.*

Returns the result of the nameservice lookup, note that either *ip* or *name* must be set.

nameservice_setlock

Parameter	Type	Req	*Description*
ip	string	*	*ip-address to lock/unlock.*
lock	number		*locks (1) or unlocks (0) the name-ip mapping.*
ips	*array*		*Array of ip-addresses to lock/unlock*

This command expects *ip* or *ips* to be set.

nameservice_update

Parameter	Type	Req	*Description*
ip	string		*ip-address to modify.*
name	string	*	*name to be used in the name-resolution.*
lock	*number*		*locked (1) or unlocked(0)*

This command expects *ip* or *ips* to be set.

note_attach

Parameter	Type	Req	*Description*
note_id	number	*	*id of existing note to attach to alarm (nimid), or zero (0) if a create+attach is performed.*
nimid	string	*	*alarm message-id that we want to attach note to*
description	string		*note description (if create)*
body	string		*note body (if create)*
category	string		*note category (if create)*
nimids	*array*		*a table of alarm message-ids*

This command primarily attaches an existing note to one or more alarms. However, it can also perform a “create and attach” in a single operation. Specify this operation by setting note_id = 0 (zero).

note_create

Parameter	Type	Req	*Description*

note_id	number	*	*id of existing note (if edit) or zero (0) if a new note is created.*
description	string	*	*note description*
body	string		*note body*
category	string		*note category*
autoremove	*number*		*auto-remove when last alarm reference is cleared.*

note_delete

Parameter	**Type**	**Req**	***Description***
note_id	*number*	*	*note id to delete*

note_detach

Parameter	**Type**	**Req**	***Description***
note_id	number	*	*id of note to remove from the alarm.*
nimid	string	*	*alarm message-id that we want to remove note from.*
nimids	*array*		*a table of alarm message-ids*

note_list

Parameter	**Type**	**Req**	***Description***
nimid	*string*		*alarm message-id that we list notes for.*

reorganize

Parameter	**Type**	**Req**	***Description***
by	*string*	*	*specifies who requested the database reorganize.*

The reorganize command will take the NAS into maintenance mode, stopping all service modules and performs a VACUUM of the *database.db* and the *transactionlog.db.* All services are started upon completion.

repl_queue_post

Parameter	**Type**	**Req**	***Description***
name	*string*	*	*specifies which NAS posts replication data.*

This is a private interface used by the NAS replication service.

repl_queue_info

Parameter	**Type**	**Req**	***Description***
name	*string*		*specifies which queue to get information about.*

script_delete

Parameter	**Type**	**Req**	***Description***
name	*string*	*	*specifies the script to delete (including directory path)*

script_rename

Parameter	**Type**	**Req**	***Description***
directory	string	*	*specifies the path where the scripts recides.*
from	string	*	*old name*
to	*string*	*	*new name*

script_list

Returns a PDS containing a string table with all scripts (including path), as well as the actual script root directory.

script_run

Parameter	**Type**	**Req**	***Description***
name	string	*	*specifies the script run*
profile	*string*		*if script is to be executed in a AO-profile context*

script_validate

Parameter	**Type**	**Req**	***Description***
name	string	*	*specifies the script run*
profile	string		*if script is to be validated in a ao-profile context*
code	string	*	*lua code to be validated*
evaluate	*number*		*get returnvalue from script.*

set_alarm

Parameter	**Type**	**Req**	***Description***
token	string	*	*specifies the element to change*
value	string	*	*Specifies a value for the token*
nimid	string	*	*alarm message-id*
nimids	*array*		*a string table of message-ids*

Key can currently be *visible (yes or no) orcustom_1-5 (value).* Either *nimid* or *nimids* is required.

set_loglevel

Parameter	**Type**	**Req**	***Description***
level	*number*	*	*specifies the nas loglevel.*

set_visible

Parameter	**Type**	**Req**	***Description***
visible	number	*	*sets the alarms visible (1) or invisible(0)*
nimid	string	*	*alarm message-id*
nimids	*array*		*a string table of message-ids*

Either *nimid* or *nimids* is required.

transaction_list

Parameter	**Type**	**Req**	***Description***
mode	string	*	*one of: today, lasthour, last24hours, last3days, lastmonth and date=ISO-startdate,ISO-enddate. E.g. date=2007-08-24,2007-08-27*
where	string		*valid SQL-92 conformant WHERE clause.*
nimid	*string*		*alarm message-id*

Returns a list of alarms that occurred in the period specified by *mode*. If *nimid* is specified then the events for that particular alarm-id are listed.

trigger_list

Parameter	Type	Req	*Description*
name	string		*name of trigger to list*
detail	*string*		*detail level, no-detail is zero(0), show events is 1.*

Troubleshooting

Large databases

Problem description
Database files are large even though no/few alarms are present in NAS.

Solution
The transaction-log database files should normally be compressed through its configured housekeeping routines. You may, however, run the advanced command 'Reorganize database' from the status action menu. This will take the NAS services offline, and compress/reorganize the databases (both database.db and the transactionlog.db). The recommended file size is <= 1 GB.

Database seems locked

Problem description
The logfile will reveal traceinformation regarding NAS subprocesses being unable to insert data into the database tables. A db-journal file older than one day is present.

Solution
The <database>.db-journal file contains the uncomitted transactions. This can be removed to restore the state of the database.

Need to browse the embedded databases

Problem description
Can we access the databases from other applications for reporting, housekeeping etc.?

Solution
Yes, there are many database management applications for the SQLite3 database. You'll find one at www.sqlite.org or by requesting one from Nimsoft support.

Old transactions are not being copied

Problem description
The transaction database is not populated with the transactions found in the old logs/ directory.

Solution
The setup/transaction_filter controlled what to store in the transaction-logfiles prior toversion 3. When the nas attempts to convert the files it expects a 'open' transaction prior to storing any information related to a message. In other words if the user configured the old nas to only store suppression and assign, then the transaction_filter=12 and no transactions will be converted. The default setting of the transaction_filter was 15. (open,close,suppress,assign)

Reset User-Interface views

Problem description
User wants to reset all NAS UI settings

Solution
All settings are stored in registry under the following key:

HKEY_CURRENT_USER\Software\VB and VBA Program Settings\conf_nas\

Appendix A – The NAS – LUA extensions

Alarm

alarm.get ([NimId])
Returns a table of alarm data for the given *nimid*. If used without the *nimid* it will return the alarm data, and is only used in conjunction with AO profiles.

alarm.list ([Field, Value [, Value…]])
Returns an array of table elements containing alarm data. Will, if used with the field and value(s) parameters, filter the result set according to the user criteria. Use the column name for your field and one or more match strings. The % is used as the wildcard character. E.g alarm.list ("hostname","%xp%") returns alarms for all hostnames with 'xp' in them. These records are extracted from the NAS_ALARMS table.

alarm.transactions (NimId)
Returns an array of table elements containing alarm transaction information for the given nimid. These records are extracted from the NAS_TRANSACTION_LOG table, in the transationlog database.

alarm.statistics ([ShowAll [, Field, Value]])
Returns a table containing the following items:

level_information	- number of informational alarms.			
level_warning	-	-"-	warnings	-"-
level_minor	-	-"-	minor	-"-
level_major	-	-"-	major	-"-
level_critical	-	-"-	critial	-"-
alarm_count	- total number of alarms.			

ShowAll = **true** will list all visible and invisible alarms. You may use the *Field* and *Value* paramters to selectively choose statistics. You may choose one of *origin, hostname, source, subsystem, sid.*

alarm.history (Selector [, Option])
Returns an array of table elements containing alarm summary records for the selected time period. The valid selectors are the ones found in the *history browser*, namely: *today, lasthour, lastweek, lastmonth,last24hours,last7days,last3days, date*. The Option parameter when used with one of the selectors mentioned, can be one of *time, closed* or *created.* The Option parameter when used with the selector "where" is a valid SQL WHERE statement (without the WHERE).
The *date* selector is on the form *date,yyyy-mm-dd [HH [:MM [:SS]]] ,yyyy-mm-dd [HH [:MM [:SS]]]*
E.g `alarm.history ("date,2007-10-18,2007-10-22 08:00")`

alarm.query (SQL-Query [, Token])
Runs the SQL-Query in the NAS database unless Token is specfied. Token may currently be "transactionlog". Returns the result of the SQL-Query. Please take causion in using this function. No checks are performed, and the caller may wreck the database or database-model by running queries.

alarm.set (AlarmTable)
Updates the existing alarm denoted by the *nimid* element of the *AlarmTable* with a set of supported fields in the alarm. The fields are *message, level* or *severity,prevlevel, sid, user_tag1, user_tag2, custom_1 to custom_5, visible* and *escalated.* The *visible* and *escalated* take 0 (false) and 1 (true) as values. The message, severity, sid, user_tag1,user_tag2 take strings as values. *Please note that this function should not be used directly with data returned by alarm.get (). Use this function with caution, especially when used in on_arrival AO profiles to avoid deadlock situations (e.g. modifying a message that rematches the filter).*

E.g Making an existing alarm invisible and setting the severity level.

```
b = alarm.get("KG12271949-57003")

a = {}
a.nimid     = b.nimid
a.level     = NIML_INFORMATION
a.prevlevel = b.level
a.visible   = 0

alarm.set(a)
```

Database

database.open ([FileName | ConnectionString , [BreakOnError]])

Opens a database handle to the specified file or database. Subsequent *database* operations will now be reference through this handle, until it is closed using the *database.close* or through an implisit close when opening another database using *database.open.* Set BreakOnError to false if you want to catch the error instead of letting the script halt. The default database is called *user.db* , See examples below:

Opens a separate SQLite database file:

```
database.open("my_private.db")
```

Opens the NiS:

```
database.open("provider=nis;database=nis;driver=none")
```

Opens MS Access database:

```
database.open ("Driver={Microsoft Access Driver
(*.mdb)};Dbq=c:\\ myluadb.mdb;Uid=xxx;Pwd=yyy")
```

database.query (SQL-Query)

Performs the provided SQL in the current open database. If no previous *database.open* has been performed then the *user.db* is used. The SQL statement must be supported by the underlying database. This function returns *tRecordSet [, iReturnCode [,sError]].* If no record-set is returned by the query then an empty table and the returncode NIME_NOENT is returned.

database.close ( )

Closes the current database.

database.setvariable (Name, Value)

Creates (or modifies) the persistent variable *Name* in the current database. The variable name should be a unique name to avoid collisions.

database.getvariable (Name)

Retrieves the persistent variable *Name.* The function returns *nil* when the variable is non-existent.

Action

action.assign (AssignTo, NimId | NimId-List [, AssignedBy])

Assigns a user to one or more alarms, using the *nimid (* or a comma separated list of *nimids*).

action.close (NimId | NimId-List)

Closes the open alarm referenced by the single *nimid* or the list of alarm ids.

action.command (CommandLine)

Executes the provided command-line string, and places the output (if any) into a table of lines. The exit-code can is returned as the second output parameter. E.g output,rc = action.command (“ls –al”)

action.note (NoteName, NoteDescription, NimId [, Overwrite])

Create and attach a note to the alarm message referenced by the *nimid*.

action.ping (HostName [, Timeout])

Returns the status (true or false) and the time-used (in milliseconds) when issuing a ping (ICMP ECHO) to the provided hostname or ip-address.

action.email (ReceiverAddress, Subject [, Body]])

Generates an email-message targeted for the NimBUS Email Gateway.

action.SMS (PhoneNumber, MessageText)

Generates an SMSI-message targeted for the NimBUS SMS Gateway.

action.profile ([Name, RunState [,Persistent]])

Activates or deactivates the named Auto-Operator profile. A persistent change will affect the configuration file. The variables RunState and Persistent are of type Boolean (true or false). Note that action.profile() returns a table of all AO-profiles with status information (see example below)

```
out = action.profile()
if out ~= nil then
   printf ("Number of profiles: %d",out.num_profiles)
   for i,f in pairs(out.profiles) do
      printf ("name: %s, active: %s, type: %s",f.name,f.active,f.type )
   end
end
```

action.filter ([Name, RunState [,Persistent]])

Activates or deactivates the named Auto-Operator pre-processing filter. A persistent change will affect the configuration file. The variables RunState and Persistent are of type Boolean (true or false). Note that action.filter() returns a table of all filters with status information (see example)

```
out = action.filter()

if out ~= nil then
   printf ("Number of filters: %d",out.num_filters)
   for i,f in pairs(out.filters) do
      printf ("name: %s, active: %s, type: %s",f.name,f.active,f.type )
   end
end
```

action.log (Activity [, Status [,TimeUsed [,Module [, Identifier]]]])

Adds activity information to the activity logger. Describe it as precise as possible, and use the status information to flag different states, or results of operations.

action.visibility (Visible, NimId | NimID-List)

Set alarm visibility to **true** or **false** on one or more alarms.

action.escalate (SeverityLevel, NimId | NimID-List)

Raises the severity level to according to the SeverityLevel parameter. Only alarms with a current severity level lower than *SeverityLevel* will be modified.

Nimbus

nimbus.alarm (SeverityLevel, MessageText [, SuppressionKey [, SubsystemId [, Source]]])

Generates a NimBUS alarm message with the severity level (1-5) and a message-text. Use the suppression-key to create a stateful alarm. Returns a return code and the message-id string.
E.g. rc,nimid = nimbus.alarm (NIML_WARNING, “help me..”)

nimbus.post (Subject, PDSHandle)

Posts a NimBUS Message onto the NimBUS using the Subject.
Returns a message-id string if successful or **nil**.

nimbus.request (NimBUSAddress, Command, Arguments [, Wait [, ReturnAsPDS]])

Returns the result of the command targeted for the provided nimbus component. The command-arguments are expected to be a PDS (returned by pds.create). The result is placed into a table unless the ReturnAsPDS parameter is set to *true*.
Please note that this is an associative table (not indexed), meaning that a PDS sections will be referenced by its section-name.

controller = nimbus.request (“controller”,”get_info”)
printf (“controller robot: %s”, controller.robotname)

nimbus.qos_definition (QosName, QosGroup, Description, Unit, UnitAbbreviation, HasMax [, IsAsynch])

Creates a QoS definition named *QosName*. Unless the flag IsAsynch is *true*, an interval based QoS is created. Please note that subsequent definitions on the same name will not recreate or alter an existing QoS definition. The HasMax flag set requires that all qos data (issued by *nimbus.qos*) referring to this *QoSName* is issued with a MaxValue.

nimbus.qos (QosName, Source, Target, Value, Interval | QOS_ASYNCH [,MaxValue])
Will send an interval based QoS message when *Interval* is greater than zero, and a asynchronous QoS message when called with QOS_ASYNCH. Please note that no QoS data will be recorded unless a valid QoS definition has been sent prior to this request. Remember to set the MaxValue if definition was created using HasMax=true.

nimbus.session_open (NimBUSAddress)
Opens a session to the targeted NimBUS component. Returns a handle to the session.

nimbus.session_request (SessionHandle, Command [, Arguments [, Wait [, ReturnAsPDS]]])
See nimbus.request.

nimbus.session_close (SessionHandle)
Closes and removes the data structure associated with the handle.

Note

note.create (Name, Description [,AutoRemove [,Category]])
Creates a note with the provided name and description fields set, and returns the note identification number (NoteId).

note.append (Name | NoteId, Description [, Overwrite])
Appends the descriptive text to an existing note defined by the name or the id. A new note will be created when no matches are found. Returns status.

note.delete (Name | NoteId)
Deletes a note with the provided name or id. Returns status.

note.find (Name)
Returns the NoteId of the named note, and the note description as the second return parameter, or **nil** when nothing matches the provided Name.

note.attach (Name | NoteId, NimId [, NimId…])
Attaches the note to one or more alarms specified as NimIds.

Trigger

trigger.alarms (TriggerName)
Returns an array of table elements containing alarm data matching the criteria for the named trigger.

trigger.count (TriggerName)
Returns the number of alarm events currently matching the trigger criteria.

trigger.exist (TriggerName)
Returns *true* if the trigger is defined.

trigger.list ([TriggerName])
Returns a table of triggers, or the table entry for the matching trigger name.

trigger.state (TriggerName)
Returns the state (raised or not raised) of the named trigger.

trigger.timestamp (TriggerName)
Returns the UTC timestamp when the trigger last changed state.

File

file.copy (Source, Destination)

Creates a file using the complete *Path* and writes *Buffer* into the file if provided.

file.create (Path [, Buffer])
Creates a file using the complete *Path* and writes *Buffer* into the file if provided.

file.delete (Path)
Deletes the file named *Path.*

file.read (Path [,Mode [,StartPos]])
Returns a buffer with the filecontents, and the number of bytes read as a second return parameter. The optional *mode* parameter allows for controlling the open-mode. (see fopen man-pages, default: “rb”).*Startpos* indicates the position where to read from (default: 0)

file.write (Path , Buffer)
Appends *Buffer* the file *Path*, and returns **true** if success

file.stat (Path)
Returns a table containing the following statistics: *mtime, ctime, atime, mode* and *size*.

file.rename (OldName , NewName)
Renames the file OldName to NewName.

file.checksum (Path)
Returns a Base64 encoded checksum string for the specified file.

file.list (Path [, Pattern [, DirectoriesOnly]])
Returns a string table with the filenames (or directories if the DirectoriesOnly is set to **true**). The *pattern* can be used to specify the search pattern (default is *).
E.g

```
l = file.list ("\\tmp","*.log")
for k,v in pairs(l) do
   printf("%s: %s",k,v)
end
```

file.mkdir (Path)
Creates the given directory. Please note that this does not support a recursive create.

Timestamp

timestamp.now ()
Returns the number of seconds elapsed since Jan. 1 1970, 00:00:00.

timestamp.diff (StartTimeStamp [, Format [, EndTimeStamp]])
Returns the difference (seconds, minutes,hours or days) between the EndTimeStamp (or *now* if not provided) and the StartTimeStamp using the Format specifier (**s**econds, **m**inutes, **h**ours,**d**ay)

timestamp.newer (TimeStamp, TimeSpecification)
Returns **true** if the TimeStamp is newer than specified by the TimeSpecification. The TimeSpecification format is built using a combination of numbers and the tokens: **s**econds, **m**inutes, **h**ours, **d**ays. E.g. 10h30min, 5hrs, 30m, 3 days

timestamp.older (TimeStamp, TimeSpecification)
Returns **true** if the TimeStamp is older than specified by the TimeSpecification. The TimeSpecification format is built using a combination of numbers and the tokens: **s**econds, **m**inutes, **h**ours, **d**ays. E.g. 10h30min, 5hrs, 30m, 3 days

timestamp.data ([TimeStamp])
Uses ‘*now*’ if no parameter is provided. Returns a table with the following self-explanatory members: *year,month,day,hour,minute,second,yearofday,weekday* and *isdst* (1 if daylight savings time).

timestamp.fromISO (ISOdatestring)
Returns a timestamp and a timestamp data table (see *timestamp.data*).

timestamp.format (TimeStamp [, Format])
Returns a formatted timestring using the Format specifier (default: %b %d, %H:%M:%S).

specifier	Replaced by	Example
`%a`	Abbreviated weekday name *	`Thu`
`%A`	Full weekday name *	`Thursday`
`%b`	Abbreviated month name *	`Aug`
`%B`	Full month name *	`August`
`%c`	Date and time representation *	`Thu Aug 23 14:55:02 2001`
`%d`	Day of the month (`01-31`)	`23`
`%H`	Hour in 24h format (`00-23`)	`14`
`%I`	Hour in 12h format (`01-12`)	`02`
`%j`	Day of the year (`001-366`)	`235`
`%m`	Month as a decimal number (`01-12`)	`08`
`%M`	Minute (`00-59`)	`55`
`%p`	AM or PM designation	`PM`
`%S`	Second (`00-61`)	`02`
`%U`	Week number with the first Sunday as the first day of week one (`00-53`)	`33`
`%w`	Weekday as a decimal number with Sunday as `0` (`0-6`)	`4`
`%W`	Week number with the first Monday as the first day of week one (`00-53`)	`34`
`%x`	Date representation *	`08/23/01`
`%X`	Time representation *	`14:55:02`
`%y`	Year, last two digits (`00-99`)	`01`
`%Y`	Year	`2001`
`%Z`	Timezone name or abbreviation	`CDT`
`%%`	A `%` sign	`%`

* The specifiers whose description is marked with an asterisk (*) are locale-dependent.

PDS

The PDS (Portable Data Stream) format is used heavily within the NimBUS to exchange data between various processes on all platforms supported by NimBUS. This format allows users to build nested datastructures that may be passed between different languages and different hardware platforms.

pds.create ()
Returns a reference handle to a PDS structure. Use **pds.size** (pdsHandle) for size information.

pds.copy (pdsHandle)
Returns a reference handle to a copied PDS structure.

pds.delete (pdsHandle)
Deletes the PDS structure and data.

pds.convert (pdsHandle)
Returns a LUA table. This function converts the PDS structure to a LUA table containing the same key/value pairs and sub-tables (if any).

pds.putInt (pdsHandle, Key, Value)
Stores an integer value in the provided PDS structure using the Key as the reference to the Value.
Note that an existing element with the same Key will be replaced.

pds.putString (pdsHandle, Key, Value)
Stores a string in the provided PDS structure using the Key as the reference to the Value. Note that an existing element with the same Key will be replaced.

pds.putDouble (pdsHandle, Key, Value)
Stores a double value in the provided PDS structure using the Key as the reference to the Value. Note that an existing element with the same Key will be replaced.

pds.putPDS (pdsHandle, Key, Value)
Stores a PDS in the provided PDS structure using the Key as the reference to the Value.
Note that an existing element with the same Key will be replaced.

pds.putTable (pdsHandle, Key, Value)
Stores a value of type *string, number* or *PDS* to the named table *Key*.

pds.getInt (pdsHandle, Key)
Returns the number associated by Key from the provided PDS structure (or **nil** if non-existent).

pds.getString (pdsHandle, Key)
Returns the string value associated by Key from the provided PDS structure (or **nil** if non-existent).

pds.getDouble(pdsHandle, Key)
Returns the number associated by Key from the provided PDS structure (or **nil** if non-existent).

pds.getPDS (pdsHandle, Key)
Returns the PDS handle associated by Key from the provided PDS structure (or **nil** if non-existent).

pds.getTableInt (pdsHandle, Key, TableIndex)
Returns the number associated by Key from the named table Key and index *TableIndex*. Zero (0) is the first table index.

pds.getTableString (pdsHandle, Key, TableIndex)
Returns the string value associated by Key from the named table Key and index *TableIndex*. Zero (0) is the first table index.

pds.getTablePDS (pdsHandle, Key, TableIndex)
Returns the PDS handle associated by Key from the named table Key and index *TableIndex*. Zero (0) is the first table index.

pds.getNext (pdsHandle)
Returns the next Key, Type, DataSize, Data from the provided PDS structure (or **nil** if non-existent).

pds.size()
Returns the number of bytes.

pds.fileOpen (sPath)
Returns a reference handle to an open pdsFile. Close the file using **pds.fileClose**.

pds.fileClose (pdsFileHandle)
Closes the pdsFile.

pds.fileRead (pdsFileHande [,bMarkAsRead])
Returns 3 optional output parameters: return code, a reference handle to a PDS and the number of bytes read. The bMarkAsRead flag advances and saves the file read pointer. Note that the PDS file can contain blocks of PDS's, these can be read in a loop.
E.g `rc,dta,nbytes = pds.fileRead(f,true)`

pds.fileWrite (pdsFileHande , pdsHandle)
Writes the provided pdsHandle data to the file.

sprintf (Format [,Par1 [,Par2 […]]])
Returns a string buffer with the formatted string.

printf (Format [,Par1 [,Par2 […]]])
Logs the formatted string to the output window (if in the editor) or the NAS logfile.

print (Par1[,Par2 […]]])
Logs the string to the output window (if in the editor) or the NAS logfile. Used primarily for simple unformatted printing and debug output.

left (String, Length)
Returns *Length* characters from the *String*, starting from the left.

right (String, Length)
Returns *Length* characters from the *String*, starting from the right.

mid (String, Start [, Length])
Returns *Length* characters from the *String*, starting from *Start*. If no *Length* is specified, the rest of the string will be returned.

substr (String, Substring)
Returns **true** if the *Substring* is found, as well as the starting *Position* of the subsctring.

split (String [, Separators])
Returns a table of substrings separated by one or more of the Separator characters. The default separator is whitespace.

trim (String [, Mode])
Removes leading and/or trailing whitespaces from *String*.
The *Mode* may be 0: leading and trailing, 1: leading, 2: trailing. The default value is 0.

regexp (String, Expression)
Returns **true** if the regular (or pattern matching) expression matches *String*.

setvariable (Name, Value)
Stores the non-persistent variable *Name*. The value is retrievable until a cold-start of the NAS clears the Non-persistent data store. Use the equivalent **database.setvariable** for a persistent store.

getvariable (Name)
Returns the non-persistent named variable *Name* or **nil** if non-existent.

exit (ExitCode)
Terminates the script execution with an *ExitCode*. Non-zero *ExitCodes* will be recorded in the NAS activity-log.

tonumber (Value)
Converts *Value* into a number.

tostring (Value)
Converts *Value* into a string.

type (Value)
Returns the variable type as as string.

state (TriggerName)
This is a shortcut for the **trigger.state** function.

sleep (MilliSeconds)
Suspends execution for a given time.

Constants

NIML_CLEAR	= 0
NIML_INFORMATION	= 1
NIML_WARNING	= 2
NIML_MINOR	= 3
NIML_MAJOR	= 4
NIML_CRITICAL	= 5

NAS_ALARM_NEW	= 1
NAS_ALARM_UPDATE	= 2
NAS_ALARM_CLOSE	= 4
NAS_ALARM_ASSIGN	= 8
NAS_ALARM_ACK	= 32

NIME_OK	= Ok
NIME_AGAIN	= Not ready, try again
NIME_ERROR	= Error
NIME_COMERR	= Communication/connectivity error
NIME_INVAL	= Invalid argument
NIME_NOENT	= No such entry
NIME_ISENT	= Entry is already defined
NIME_ACCESS	= No access

QOS_ASYNCH	= -1
NAS_AO_INTERVAL	= from the current NAS configuration.
NAS_NAME	= the name of the current NAS.
NAS_ADDRESS	= the NimBUS address of the current NAS.
SCRIPT_NAME	= the name of the executing script.
SCRIPT_FILE	= the filename of the executing script.
SCRIPT_ARGUMENT	= the argument string passed from the executing AO profile.
PROFILE_NAME	= the AO profile that executed the script (if any).
PROFILE_STATE	= the state of the profile when using the *on_trigger* method.

Table structures

As returned from **alarm.list(), alarm.get()**:

.nimid	- unique NimBUS Id
.nimts	- timestamp when the alarm was created (at source)
.source	- source of the alarm (typically ip-address)
.hostname	- resolved name (robotname or ip-address to name resolution)
.level	- severity level (0-5)
.severity	- textual representation of the severity *level.*
.supptime	- timestamp of last suppression.
.sid	- subsystem identification.
.subsys	- subsystem string resolved from *sid.*
.message	- alarm message text.
.suppcount	- number of times event has been suppressed.
.supp_key	- suppression identification key.
.origin	- origin of the alarm (stamped by nearest hub, or in some cases the robot.)
.domain	- name of originating NimBUS domain.
.robot	- name of the sending robot.
.hub	- name of the nearest hub to the sending robot.
.nas	- name of originating alarm server.
.prid	- name of probe issuing the alarm.
.user_tag1	- user tag 1 (as set by robot).
.user_tag2	- user tag 2 (as set by robot).
.visible	- flag for visibility (1 = visible)
.aots	- AO timestamp
.arrival	- timestamp when alarm arrived at NAS.
.time_arrival	- datetime of *arrival.*
.time_supp	- datetime of *supptime*.
.time_origin	- datetime of *nimts.*
.assigned_at	- datetime at assignment.
.assigned_to	- user alarm is assigned to.
.assigned_by	- the user who assigned the alarm.
.tz_offset	- timezone offset (seconds from GMT)
.supp_id	- checksum of suppression information.
.change_id	- checksum of message,severity and subsystem.
.dev_id	- device_id of message originator.
.met_id	- metric_id of originator.
.custom_1	- user definable custom field (custom_1 through custom_5)

As returned by **alarm.transactions(),alarm.history()**:

.nimid	- unique NimBUS Id
.source	- source of the alarm (typically ip-address)
.hostname	- resolved name (robotname or ip-address to name resolution)
.level	- severity level (0-5)
.severity	- textual representation of the severity *level.*
.time	- datetime of event.
.sid	- subsystem identification.
.subsys	- subsystem string resolved from *sid.*
.message	- alarm message text.
.suppcount	- number of times event has been suppressed.
.origin	- origin of the alarm (stamped by nearest hub, or in some cases the robot.)
.domain	- name of originating NimBUS domain.
.robot	- name of the sending robot.
.hub	- name of the nearest hub to the sending robot.
.nas	- name of originating alarm server.
.prid	- name of probe issuing the alarm.
.user_tag1	- user tag 1 (as set by robot).
.user_tag2	- user tag 2 (as set by robot).
.visible	- flag for visibility (1 = visible)
.assigned_to	- user alarm is assigned to.
.assigned_by	- the user who assigned the alarm.
.acknowledged_by	- the user who acknowledged the alarm.
.tz_offset	- timezone offset (seconds from GMT)

.type[2]	- transaction type (New,Suppressed major/minor,Acknowledged,Assigned,Closed)
.dev_id[1]	- device_id of message originator.
.met_id[1]	- metric_id of originator.
.custom_1[1]	- user definable custom field (custom_1 through custom_5)

[1] Only returned by **alarm.history()**
[2] Only returned by **alarm.transactions()**

As returned by **alarm.statistics()**:

.level_clear	- number of open alarms with severity level *clear*.
.level_information	- number of open alarms with severity level *information*.
.level_warning	- number of open alarms with severity level *warning*.
.level_minor	- number of open alarms with severity level *minor*.
.level_major	- number of open alarms with severity level *major*.
.level_critical	- number of open alarms with severity level *critical*.
.alarm_count	- number of open alarms.
.oldest_alarm	- timestamp of the oldest open alarm.
.newest_alarm	- timestamp of the newest open alarm.

Custom Pre-Processing

The *event* table is placed into the LUA context prior to executing the "custom" pre-processing rule. You may alter (launder) the event by setting the fields *message, level, sid, source, hostname, user_tag1, user_tag2, visible, custom_1 to custom_5, supp_key* and *origin.* The following fields are present for the script to use:

.source	- source of the alarm (typically ip-address)
.hostname	- resolved name (robotname or ip-address to name resolution)
.level	- severity level (0-5)
.sid	- subsystem identification.
.message	- alarm message text.
.origin	- origin of the alarm (stamped by nearest hub, or in some cases the robot.)
.domain	- name of originating NimBUS domain.
.robot	- name of the sending robot.
.hub	- name of the nearest hub to the sending robot.
.prid	- name of probe issuing the alarm.
.user_tag1	- user tag 1 (as set by robot).
.user_tag2	- user tag 2 (as set by robot).
.supp_key	- suppression identification key.
.visible	- flag for visibility (true = visible)

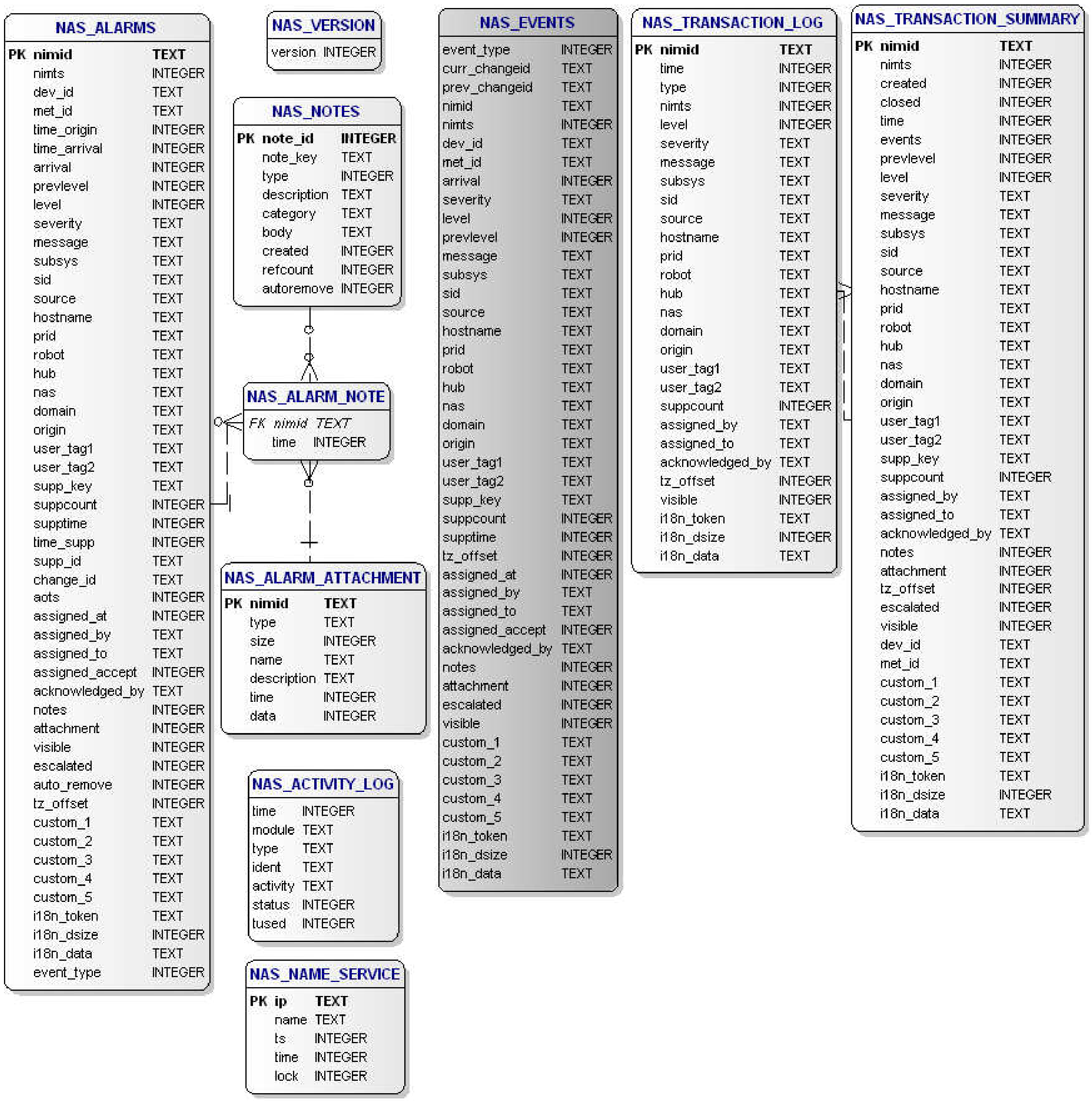
The script is expected to return the *event* (modified or not) or *nil*. The *nil* will indicate that the event is to be skipped.

Note that the *user_tag1* and *user_tag2* fields will be stored in the database when the inbound alarm translates into a *new* event. Please take care when/if modifying the supp_key, since it alters how the sending probe identifies the checkpoint.

Note that all pre-processing handling will by nature slow down the processing of inbound alarms.

Note that only a subset of the lua methods are available to the pre-processing script. The following classes and methods are not available: *exit, sleep, nimbus, pds, trigger, action, database, alarm and note.* The *trigger.state* method through the *state* method is however available.

Appendix B – Database Schemas (database.db and transactionlog.db)



NAS_ALARMS

PK	**nimid**	**TEXT**
	nimts	INTEGER
	dev_id	TEXT
	met_id	TEXT
	time_origin	INTEGER
	time_arrival	INTEGER
	arrival	INTEGER
	prevlevel	INTEGER
	level	INTEGER
	severity	TEXT
	message	TEXT
	subsys	TEXT
	sid	TEXT
	source	TEXT
	hostname	TEXT
	prid	TEXT
	robot	TEXT
	hub	TEXT
	nas	TEXT
	domain	TEXT
	origin	TEXT
	user_tag1	TEXT
	user_tag2	TEXT
	supp_key	TEXT
	suppcount	INTEGER
	supptime	INTEGER
	time_supp	INTEGER
	supp_id	TEXT
	change_id	TEXT
	aots	INTEGER
	assigned_at	INTEGER
	assigned_by	TEXT
	assigned_to	TEXT
	assigned_accept	INTEGER
	acknowledged_by	TEXT
	notes	INTEGER
	attachment	INTEGER
	visible	INTEGER
	escalated	INTEGER
	auto_remove	INTEGER
	tz_offset	INTEGER
	custom_1	TEXT
	custom_2	TEXT
	custom_3	TEXT
	custom_4	TEXT
	custom_5	TEXT
	i18n_token	TEXT
	i18n_dsize	INTEGER
	i18n_data	TEXT
	event_type	INTEGER

NAS_VERSION

version	INTEGER

NAS_NOTES

PK	**note_id**	**INTEGER**
	note_key	TEXT
	type	INTEGER
	description	TEXT
	category	TEXT
	body	TEXT
	created	INTEGER
	refcount	INTEGER
	autoremove	INTEGER

NAS_ALARM_NOTE

FK	*nimid*	*TEXT*
	time	INTEGER

NAS_ALARM_ATTACHMENT

PK	**nimid**	**TEXT**
	type	TEXT
	size	INTEGER
	name	TEXT
	description	TEXT
	time	INTEGER
	data	INTEGER

NAS_ACTIVITY_LOG

time	INTEGER
module	TEXT
type	TEXT
ident	TEXT
activity	TEXT
status	INTEGER
tused	INTEGER

NAS_NAME_SERVICE

PK	**ip**	**TEXT**
	name	TEXT
	ts	INTEGER
	time	INTEGER
	lock	INTEGER

NAS_EVENTS

event_type	INTEGER
curr_changeid	TEXT
prev_changeid	TEXT
nimid	TEXT
nimts	INTEGER
dev_id	TEXT
met_id	TEXT
arrival	INTEGER
severity	TEXT
level	INTEGER
prevlevel	INTEGER
message	TEXT
subsys	TEXT
sid	TEXT
source	TEXT
hostname	TEXT
prid	TEXT
robot	TEXT
hub	TEXT
nas	TEXT
domain	TEXT
origin	TEXT
user_tag1	TEXT
user_tag2	TEXT
supp_key	TEXT
suppcount	INTEGER
supptime	INTEGER
tz_offset	INTEGER
assigned_at	INTEGER
assigned_by	TEXT
assigned_to	TEXT
assigned_accept	INTEGER
acknowledged_by	TEXT
notes	INTEGER
attachment	INTEGER
escalated	INTEGER
visible	INTEGER
custom_1	TEXT
custom_2	TEXT
custom_3	TEXT
custom_4	TEXT
custom_5	TEXT
i18n_token	TEXT
i18n_dsize	INTEGER
i18n_data	TEXT

NAS_TRANSACTION_LOG

PK	**nimid**	**TEXT**
	time	INTEGER
	type	INTEGER
	nimts	INTEGER
	level	INTEGER
	severity	TEXT
	message	TEXT
	subsys	TEXT
	sid	TEXT
	source	TEXT
	hostname	TEXT
	prid	TEXT
	robot	TEXT
	hub	TEXT
	nas	TEXT
	domain	TEXT
	origin	TEXT
	user_tag1	TEXT
	user_tag2	TEXT
	suppcount	INTEGER
	assigned_by	TEXT
	assigned_to	TEXT
	acknowledged_by	TEXT
	tz_offset	INTEGER
	visible	INTEGER
	i18n_token	TEXT
	i18n_dsize	INTEGER
	i18n_data	TEXT

NAS_TRANSACTION_SUMMARY

PK	**nimid**	**TEXT**
	nimts	INTEGER
	created	INTEGER
	closed	INTEGER
	time	INTEGER
	events	INTEGER
	prevlevel	INTEGER
	level	INTEGER
	severity	TEXT
	message	TEXT
	subsys	TEXT
	sid	TEXT
	source	TEXT
	hostname	TEXT
	prid	TEXT
	robot	TEXT
	hub	TEXT
	nas	TEXT
	domain	TEXT
	origin	TEXT
	user_tag1	TEXT
	user_tag2	TEXT
	supp_key	TEXT
	suppcount	INTEGER
	assigned_by	TEXT
	assigned_to	TEXT
	acknowledged_by	TEXT
	notes	INTEGER
	attachment	INTEGER
	tz_offset	INTEGER
	escalated	INTEGER
	visible	INTEGER
	dev_id	TEXT
	met_id	TEXT
	custom_1	TEXT
	custom_2	TEXT
	custom_3	TEXT
	custom_4	TEXT
	custom_5	TEXT
	i18n_token	TEXT
	i18n_dsize	INTEGER
	i18n_data	TEXT

Note that the NAS_EVENTS table is purely for internal NAS usage and should not be used by any external application/query.