eEMU Manual

Author: Jarra Voleynik

Table of Contents

1 Installation

eEMU comes as a tar archive. The archive contains the following components:

- emu (eEMU server)
- emucleaner (eEMU message cleaner)
- setup (setup script)
- manual.ps (eEMU manual)
- emsg1.c (emsg1 source code)
- emsg1_aix.c (emsg1 source code for AIX)
- eb (ASCII eEMU browser)

Before eEMU is installed and configured, it is important to select a UID and GID to install and run it under. emu, emucleaner and eb use a port number as a parameter. This port number is a key to the configuration file for that particular server. The standard location of the configuration file is /usr/local/emu/conf/<port>.cfg. Remember that there can be multiple servers running on a single system. The Directory hierarchy section suggests a directory hierarchy that simplifies running eEMU in a multi-server configuration. For automatic eEMU startup, a simple startup script is provided. Make sure the system that runs eEMU has perl of at least 5.002 version installed.

The eEMU kit comes with a setup script. This script can be used for an initial eEMU setup; but also it works well as a configuration file generator if more than one eEMU servers need to be run on a single system.

1.1 Deciding on UID

The first thing to decide is the UID under which EMU will run. It can be **root** if message actions need to be taken and run as root. If only messaging and paging will used, an unprivileged user called **emu** will do.
If you decide on the user emu, create its account and make /usr/local/emu its home directory.

1.2 Deciding on GID

eEMU should be installed under GID of emu. The emu user has the group emu as its primary group. The /usr/local/emu/conf/<port>.cfg file contains the eEMU password. Only users in the emu group should have access to the password, therefore make sure that /usr/local/emu/conf/<port>.cfg is "chmod 750" (readable for the owner and group owner only) and "chgrp emu" (group membership set to emu).

1.3 Directory hierarchy

eEMU programs get their configuration from /usr/local/emu/conf/<port>.cfg. This configuration file specifies the location of log files, databases etc. A recommended hierarchy of subdirectories under /usr/local/emu is as follows :

- /usr/local/emu/conf/<port>.cfg (configuration file for eEMU on port <port>)
- /usr/local/emu/bin (emu, emucleaner, eb, xeb etc.)
- /usr/local/emu/emsg (emsg1 binaries for various platforms)
- /usr/local/emu/<port>/db (database directory for eEMU on port <port>)
- /usr/local/emu/<port>/logs (log files directory for eEMU on port <port>)
- /usr/local/emu/<port>/scripts (action scripts directory for input, delete and output for eEMU on port <port>)

1.4 Configuration file <port>.cfg

eEMU servers on the same system are distinguished by the port number they listen on. The principal configuration file is called /usr/local/emu/conf/<port>.cfg. ***All*** the following parameters in the configuration file ***must be set***:

a command to send messages to the eEMU manager. This is used by emucleaner
and eb/xeb. Change localhost to the node name the emu server is running on
emsgcmd /usr/local/bin/emsg1 -n localhost

password for sending messages to the emu server
password icecream

output action script, set it to null if no action script
output_script null

delete action script, set it to null if no action script
delete_script null

input action script, set it to null if no action script
input_script /usr/local/emu/2345/scripts/input.sh

directory where log files are stored. Each received message will create
an entry in a log file
logdir /usr/local/emu/2345/logs

the DBM file name for the eEMU database
dbfile /usr/local/emu/2345/db/db

interval (in seconds) at which the eEMU browser scans for new messages
binterval 10

interval (in seconds) at which emucleaner scans for expired messages.
Must be less than 60
cinterval 20

time (in seconds) the emucleaner waits before it starts its activity.
this will prevent premature message expiration on system reboots
cbootwait 420

The **setup** program that comes with the installation kit generates eEMU configuration files for you.

1.5 Setup script

eEMU comes with a setup script. It prompts you for most options. At the end it copies emu, emucleaner and eb to a specified directory and creates an initial configuration file for a specified TCP/IP port number. The setup script can also be used later on for creation of a new configuration file.

1.6 Startup script

Depending on your platform, the startup scripts should be stored in the appropriate location. The following is an example startup script. It assumes the eEMU programs are stored in /usr/local/emu/bin

```
#!/bin/ksh
EMUBINPATH=/usr/local/emu/bin
PORT=2345
EMUPATH=/usr/local/emu/$PORT
EMULOGPATH=/usr/local/emu/$PORT/logs
PASSW=icecream

case "$1" in
’start’)
      if [ -f $EMUBINPATH/emu -a $EMUBINPATH/emucleaner ]; then
            echo "EMU start"
            /usr/bin/nohup /usr/bin/su - emu -c "$EMUBINPATH/emu $PORT" \
2>&1 1>$EMUPATH/emu.log &
            sleep 3
            echo "EMU cleaner start"
        /usr/bin/nohup /usr/bin/su - emu -c "$EMUBINPATH/emucleaner $PORT" \
2>&1 1>$EMUPATH/emu.log &
      else
         echo "$EMUBINPATH/emu or $EMUBINPATH/emucleaner does not exist"
      fi
      ;;
’stop’)
            echo "EMU stop"
            /usr/local/bin/emsg1 -o suspend -n localhost -p $PORT -w $PASSW
-m "10"
            sleep 3
            kill ‘/sbin/cat $EMULOGPATH/emu.pid‘
            kill ‘/sbin/cat $EMULOGPATH/emucleaner.pid‘
      ;;
*)
      echo "usage: $0 {start|stop}"
        ;;
  esac
```

NOTE: Notice the suspend message sent before the processes are killed. It is to prevent killing eEMU while it is updating its database. The emu.log file, created in the base directory (/usr/local/emu/<port>) for a particular server, captures some error messages and warnings from the emu server, such as syntax errors in action scripts.

1.7 Compiling emsg1.c

emsg1.c should compile on most platforms supporting BSD sockets. Due to a few differences on AIX, a separate source code file called emsg1_aix.c is provided. To compile emsg1, simply enter:
$cc -o emsg1 emsg1.c
After it is done, copy emsg1 to /usr/local/emu/emsg/emsg1 and create a symbolic link called /usr/local/bin/emsg1 to it, such as:

$*ln -s /usr/local/emu/emsg/emsg1 /usr/local/bin/emsg1*

NOTE: on some platforms, such as Solaris, use the cc compiler in /usr/ucb

2 Architecture

Just like every event management system, eEMU has a manager and agents. Agents monitor resources and send messages to the manager that maintains status of a monitored resource. eEMU was designed so that agents can be simple scripts. As a result, everyone can actively participate in agent development if he wishes so. An important point to make is that eEMU agents are not SNMP based. The authors don't see SNMP a suitable protocol for monitoring operating system and application resources.

EMU consists of three components:

emu - eEMU server
emucleaner - eEMU cleaner process
eb - eEMU browser

2.1 emu - the eEMU server

emu listenes on a specified port. To start emu on port 2345, type in "emu 2345". Incoming messages are put in a database. emu handles all manipulation of messages in the database, that is addition, updates, deletion etc. This concept is deliberate in that emu can receive delete messages from a remote eEMU system.

2.2 emucleaner - the eEMU cleaner process

emucleaner takes care of message expirations. It scans the eEMU database at regular intervals (cinterval in <port>.cfg), but doesn't make any changes to it. Rather, it uses emsg1 to instruct the emu server to delete the message.

2.3 eb - the eEMU browser/console

eb is a simple ASCII based EMU browser/console. It allows to view event messages, delete/acknowledge them, annotate existing messages and send a new message.

3 emsg1

emsg1 is the agent part of emu. It is used inside agent scripts to send a message to the emu server. emsg1 uses TCP/IP sockets. Options provided for emsg1 allow to specify all the necessary information to send a specific event. emsg1 returns a value of 0 if it succeeded in sending a message, otherwise it sends a value of 1. To provide for the EMU server being busy or a network glitch, emsg1 makes up to 7 attempts to connect to eEMU. These attempts are spaced out geometrically in order to minimise a possible collision. If, after 7 attempts, emsg1 still fails to send a message, a return value of 1 is produced. All the 7 attempts take around 2 minutes.

There are many types of messages, but the following are the most important and most commonly used:

normal - lifetime of this message in the emu database is subject to a specified time-to-live expiry
delete - deletes a specified message from the eEMU database
comment - annotates and existing message
suspend - suspends the emu server processing for a specified number of seconds

emsg1 has the following syntax:

normal message

$emsg1 [-h <hostname>] [-u <user>] -o normal -n <emu server> -p port -t <time-to-live> -s <severity> \
-w <password> -c <class> -m "<message>"

If the -o option is not specified, it defaults to normal.

E.g. an alarm message from host dumbo.company.com.au about object ID=/usr/local getting full
$emsg1 -o normal -n emuserver -p 2345 -t 6m -s 1 -c /OS/UNIX/FS -w icecream -m "/usr/local is 90% full"

comment message

$emsg1 -o comment -n <emu server> -p <port> -w <password> -m "<hostname>:<object ID> comment ....."

E.g. to send an annotation ("Adminstrator notified") to the previous message about /usr/local getting full on dumbo.company.com.au
$emsg1 -o comment -n emuserver -p 2345 -w icecream -m "dumbo.company.com.au:/usr/local Administrator notified"

delete message

$emsg1 -o delete -n <emu server> -p <port> -w <password> -m "<hostname>:<object ID>"

E.g. to delete a message about object ID=/usr/local that was received earlier
$emsg1 -o delete -n emuserver -p 2345 -w icecream -m "dumbo.company.com.au:/usr/local"

suspend message

$emsg1 -o suspend -n <emu server> -p <port> -w <password> -m <seconds>

E.g. suspend processing in emuserver for 5 seconds to facilitate a clean shutdown
$emsg1 -o suspend -n emuserver -p 2345 -w icecream -m 5

hostname - the host name running "emu", if not specified with the -h option, it is supplied by emsg1. The -h option allows to override the hostname if forwarding *messages* to another emu server.
port - port number "emu" is listening on
time-to-live - lifetime of the message. It can be specifieds in seconds (e.g. 23s), minutes (e.g. 15m), hours (e.g. 4h) or as a specific time in a 24-hour interval (e.g. 14:30). It can also be set to -1 or 0. See below for details.
severity - severity of the message; a small integer (e.g. 4)
password - the emu server password. If the password is incorrect, messages are discarded
class - class of the message for easy classification of resources. E.g. OS/FS for filesystem, OS/PRO for processes, APP/DB for databases etc. The class can be used to hierarchically build classes of resources and group them.
message - message itself in quotes, resource ID or a number of seconds depending on the message type.

A simple excerpt from a process agent can look like the following:

```
TMP=/agents/tmp/ps.out
PROCESS=sendmail
ps axw >$TMP
grep $PROCESS $TMP
if [  $? -ne 0 ];then
      emsg1 -o normal -n $EMUSERVER -p 2345 -t 7m -s 1 -c /OS/UNIX/PRO -m "$PROCESS process is missing"
fi
```

Here, the agent is run from cron every 5 minutes, therefore time-to-live is set to 7 minutes. $PROCESS is the unique object ID.

4 Message "time-to-live"

Time-to-live determines how long the message will stay in the database. If no refresh of the same message arrives within time-to-live for the message, it is deleted. It is to presume that the problem reported by the agent has been fixed. Time-to-live will be predominantly determined by the polling interval of the monitored resource (actually time-to-live should be slightly greater [120 sec] than the polling interval). For some messages, such as a batch job or backup, there is no periodicity inherent to it. Such messages have time-to-live set to -1, which indicates infinity. Such a message has to be manually deleted from the event browser (eb or xeb).

In some cases, there is no need to store the message in a database. All that is needed is trip an action in the output script. Such messages have time-to-live set to 0.

One may ask what happens if the cron process running agents dies. Or, the whole system is down. To protect against loosing agents, implement a heart-beat concept. Write a heart-beat agent that sends a notification message every 5 minutes. This message will have time-to-live set to 0. The emu output action script will update a heart-beat file for the respective system on receipt of a heart-beat message. A cron job on the emu server node may run every 5 minutes to check if the heart-beat file for each system has been updated. If a heart-beat file hasn't been updated for more than 7 minutes, we know there is a problem with the respective system.

5 Message actions

Most event managers allow actions to be taken on messages. They integrate languages to achieve that. However, some are too simple and awkward to use, others to complex to learn. EMU gives you the choice of what language to use. For each received message a specified script is called. The message attributes are passed to it via environment variables. It is the script's responsibility to branch to action code based on message attributes. The script can call other scripts based on a particular message type etc. In order to reduce overhead of message scans, the invocation of the script is done in the background so that "emu" is not affected in terms of performance. Feel free to use Perl, Python, C, ksh or whichever is your favorite coding tool.

EMU uses three action scripts:

i**nput script** - is called immediatelly on receipt of a message. If the input script returns a return value greater than 0, the message is discarded. This feature can be used to implement calendars etc. Also, using the E_RHOST environment variable, the input script can enhance the security by allowing messages only from certain node names.

delete script - is called on receipt of a delete message. Can be used to synchronize eEMU with an eEMU higher in the hierarchy or a third party system.

output script - here is where most action processing takes place. Actions are invoked based on message attributes.

list of environment variables for message attributes:

E_MSGNUM	(message number)
E_TYPE	(message type - normal, delete, comment, suspend etc)
E_TIME	(time the message was sent the first time)
E_USER	(user who sent the message)
E_SEV	(severity of the event)
E_COUNT	(how many times has the same message been refreshed, new messages have E_COUNT set to 1)
E_CLASS	(message class)
E_TTL	(time to live)
E_HOST	(host name of the system that sent the message, can be overidden with the -h option for emsg1)
E_RHOST	(as the sending node name can be changed in emsg1 with the -h option, E_RHOST is the real node name as reported by the socket connection)
E_MSG	(message itself)
E_CHANGED	(if set to 1, the message text has changed for an existing message. Say the message "/usr is 92% full" has changed to "/usr is 95% full" for a system)
E_SEV_CHANGED	(if set to 1, the severity for an existing message has changed)
E_CLASS_CHANGED	(if set to 1, the class for an existing message has changed)
E_COMMENT	(contains the comment text of a comment message)
E_TXT_WRITTEN	(set to 1 if the emu server updated the text file for eEMU browser; *.txt file in the database directory. It can be used for profile text file generation - see below)
E_KEY	(message key = <hostname>:<object ID>)

A simple action script can be as follows:

```
PORT=2345
PASSWD=icecream

if [ $E_MSG = "sendmail processes is missing" ];then
   ssh $E_HOST "sendmail -bd"
   if [ $? -ne 0 ];then
      emsg1 -h $E_HOST -n localhost -p ${PORT} -t -1 -s 2 -w ${PASSWD} -c /OS/UNIX/PRO -m "sendmail restart \
failed"
   fi
fi
```

6 Log files

eEMU maintains a log file of all received messages. The name of the log file is yyyymmdd.log and it resides in the *logdir* directory specified in <port>.cfg. After midnight, an automatic roll-over into the next day log file is facilitated. The format of log file entries is as follows:

message receive time (ddmmHHMM)
message type (normal,comment, delete)
message generation time (time the first message was sent for the specific resource ID)
sending host (hostname)
sending user
severity
message count (if message received multiple times)
message class
time to live
message text

All fields are delimited with a vertical bar.

Statistical processing of the entries should be easy enough with any spreadsheet or script.

7 eb

eb is an ascii based event browser. It runs on any terminal. Apart from the system time, it displays the message number (messages are numbered for easy ascii based manipulation), time the first message was sent, system that sent it, message class, message severity and the message itself. To keep the width of an eb window small, the hostname is stripped of the domain name.

If time-to-live is set to -1 (infinity), the message is displayed in reverse video. Removal of the message must be done with the "delete" command. The first message for a resource ID is displayed in bold so that IT staff can pick up fresh messages.

To enter command mode, type CTRL-C. A chevron prompt appears. The following commands are available:

>>d <message number>	(to delete a message)
>>q	(to exit eb)
>>m	(to send a message)
>>a <message numbe>	(to annotate a message)

"eb" gets parameters from "/usr/local/emu/conf/<port>.cfg". See above for syntax of the configuration file.

8 EMUSELECT shell environment variable

IT staff usually consists of many support groups. Each group needs to see only messages for resources falling under their resposibility. To accomplish message filtering for eEMU browser, a shell environment variable called EMUSELECT can be used. It can be set in the user's .profile if only one browser view is used. In case more groups need to be viewed by a user in separate browsers, set up a shell script for each view. Inside the script, EMUSELECT will be set.

eb and xeb use EMUSELECT as a filter for messages to be displayed. Feel free to use EMUSELECT to convert messages on the fly etc.

E.g.
EMUSELECT=" sort -t^ -k 3.4rn -k 3.1rn -k 3.7rn -k 3.10rn | grep UNIX " # to display message in reverse time order (lates ones on top)
export EMUSELECT

a script to show production system only except systems listed in the standby_nodes.txt file:

#!/usr/bin/ksh
EMUSELECT=" sort -t^ -k 3.4rn -k 3.1rn -k 3.7rn -k 3.10rn | grep -vf standby_nodes.txt grep PRD "
export EMUSELECT
eb 2345

The contents of standby_nodes.txt may look like the following:

sapserever23
webserver2

where each line contains a node name.

9 Message Guide

9.1 Resource ID

Monitored resources are designated with a unique qualifier that we call an object ID. The hostname and object ID form a resource ID:

resource ID = hostname:object ID

The resource ID uniquely identifies a resource in the enterprise. It is used as a message key in the database. Messages with the same hostname and object ID as
an existing message are considered updates, irrespective of what the message text is. As an example, let's take a /usr/local filesystem on a host called
dumbo.company.com.
The resource ID in this case is dumbo.company.com:/usr/local. In order to provide for easy segregation of the object ID, it is ALWAYS specified as the first word of the message. Make sure that agents send messages with unique object IDs in them.

Let's suppose we need to monitor a process called netscape run by user jarra or victor. How do we distinguish between the two if the object ID = netscape? In that case, another qualifier is introduced as part of the object ID. There can be multiple qualifiers separated by a percent sign. Each qualifier narrows down the object ID. For example, the above netscape process will have two watchers with the following object IDs:

- netscape%jarra
- netscape%victor

In case of log watchers monitoring log files for a pattern, a sequence number is appended as part of the object ID. For example, we can have 2 watchers for /var/log/messages. One scans for the word "volerror" the other for the pattern "disk error". The following are the object IDs:

- /var/log/messages%1
- /var/log/messages%2

(NOTE: log watcher composes a message based on the log file's contents. The actual message will tell us what is wrong, not the object ID)

In case of databases, we may go into three levels if the resource hierarchy calls for it. For example, the following is an object ID for table "SALARY" in tablespace "HR" in database "ACCOUNTS":
ACCOUNTS%HR%SALARY

Given an object ID, the message text is either a full description of a problem or it may complement the object ID to form a meaningful message.

Examples of resource ID

- Oracle database of SID = ICC_PRD, tablespaces TOOLS on host ttc3232= resource ID = ttc3232:ICC_PRD%TOOLS
- /tmp filesystem on host tcc1212.companu.com = resource ID = tcc1212.company.com.au:/tmp
- process cron on host dumbo = resource ID = dumbo:cron

Why is the concept of resource IDs important ?
If we start integrating EMU with a call logging system, we will find that calls need to be logged once only for a specified resource problem. Later messages for that resource need to update an existing call only (with the new message). An example may be a filesystem whose utilization goes up even after an alarm has been triggered. It is obvious that the message text will be changing depending on the utilization percentage. Our system needs to interpret all those messages as belonging to a single resource.

9.2 EMU text file

EMU exports its dbm database into a text file for easy processing by EMU browsers. This text file resides in the database directory and has a .txt suffix. It is updated only if:

- new message (resource ID) is added
- message text of an existing message has changed
- message severity of an existing message has changed
- message comment of an existing message has changed

Only normal messages are put in the text file since they are the only message type meant for display in the browser.

9.3 Message types

EMU has several message types in order to provide maximum flexibility and functionality. The message type determines the way the message is handled by EMU. Originally, EMU started out with normal and delete messages only. Over time, more message types were added and EMU developed its own messaging language that can be used to define very complex event scenarios. Every business has its specific event flow. This flow can be described and programmed by EMU messages to achieve full control.

The first type of messages are for database inserts or updates. They are:
normal
sleep
mask

count
event

The ***second type*** of messages manipulates an existing message addressable by a resource ID. They are:
delete
wakeup
comment
query (does a message exist)

The **th*ird type*** are command messages. They instruct EMU to do something: They are:
query (to fetch a file)
suspend

9.4 normal message

This is the most frequently used message type. It is used in filesystem, process, swap and other agents. Time-to-live is set in order to preserve the message in the EMU database across resource polls. If no refresh of the same resource ID is received during time-to-live, the message is removed (expired).

EMU processing:
call input script, add/update message, log message, call output script

Syntax:
$emsg1 [-h <hostname] [-u <user] -o normal -n <emu server> -p <port> -t <time-to-live> -s <severity> \
-w <password> -c <class> -m "<message>"

If the -o option is not specified, it defaults to normal.

Example:
an alarm message from host dumbo.company.com.au about object ID=/usr/local getting full
$emsg1 -o normal -n emuserver -p 2345 -t 6m -s 1 -c /OS/UNIX/FS -w icecream -m "/usr/local is 90% full"

Use:
Any resource monitoring, such as filesystems, processes, tablespaces, mail queues. Time-to-live is set depending on the interval at which the resource is polled. Remember that time to live must be slightly longer than the polling interval. The polling interval is typically set in cron if the agent is invoked by cron. For batch jobs or backups, set time-to-live to -1 to force the message to stay displayed until it is manually acknowledged. Note that if multiple messages arrive for the same resource ID, only the latest one is displayed.

9.5 delete message

This message is used by emucleaner to delete messages whose time is up. Nevertheless, a message can be deleted from an application script as well.

EMU processing:
call input script, log message, call delete script, delete message

Syntax:
$emsg1 -o delete -n <emu server> -p <port> -w <password> -m "<hostname>:<object ID>"

Example:
to delete a message about object ID=/usr/local that was received earlier
$emsg1 -o delete -n emuserver -p 2345 -w icecream -m "dumbo.company.com.au:/usr/local"

Use:
Delete message is self-explanatory. It deletes a message of a specified resource ID from the database. It usually means that the problem has been fixed. EMU browsers allow to send a delete message for a problem acknowledgement/fix. As we will find later, delete messages are used to stop sleep messages from exploding.

comment message

For efficient and prompt communication among all the people watching EMU messages, messages annotation is indispensable.

EMU processing:
call input script, comment specified message, log message, call output script

Syntax:
$emsg1 -o comment -n <emu server> -p <port> -w <password> -m "<hostname>:<object ID> comment ....."

Notice that the comment freely follows the resource ID.

Example:
to send an annotation ("Administrator notified") to the previous message about /usr/local getting full on dumbo.company.com.au
$emsg1 -o comment -n emuserver -p 2345 -w icecream -m "dumbo.company.com.au:/usr/local Administrator notified"

Use:
Comments are a good way to communicate among multiple people about an existing message. It can be a work request number, note about ETA etc.

9.6 sleep message

Sleep message literally sleeps in the EMU database until it is woken up by emucleaner (based on its time-to-live) or by another external event (sending the wakup message). Once woken up, a sleep message is changed to a normal message with time-to-live set to -1 (infinity). The normal message must be manually deleted. Of course the deletion can be automated through a script with a delete message.

EMU processing:
call input script, log message, call delete script, delete message

Syntax:
$emsg1 [-h <hostname>] [-u <user>] -o sleep -n <emu server> -p <port> -t <time-to-live> -s <severity> \
-w <password> -c <class> -m "<message>"

Example:
a sleep message from host dumbo.company.com.au about object ID=backup%fs. The message is sent by a filesystem backup script at the beginning of the backup. A delete message for resource ID = dumbo.company.com.au:backup%fs is sent at the end of the backup. Time-to-live is set to either a typical duration of the backup or the end time of backup window. On our case, backup must finish by 6:30 am.
$emsg1 -o sleep -n emuserver -p 2345 -t 06:30 -s 1 -c /OS/UNIX/BCK -w icecream -m "backup%fs failed or is running overtime"

Use:
As the above example suggests, sleep is usually used to notify us about a negative event in case a processing script crashes. It is a prediction message that explodes if the calling program doesn't get a chance to clean it up. This feature is very powerful for failure prediction.

9.7 wakeup message

This message is sent by emucleaner to turn a sleep message into a normal message so that it shows in the EMU browser.

EMU processing:
call input script, change sleep message into normal message, log message, call output script

Syntax:
$emsg1 -o wakeup -n <emu server> -p <port> -w <password> -m "<hostname>:<object ID>"

Example:
to wakeup a message with resource ID = dumbo.company.com.au:backup%fs that was received earlier:
$emsg1 -o wakeup -n emuserver -p 2345 -w icecream -m "dumbo.company.com.au:backup%fs"

Use:
wakeups are usually sent by emucleaner when a sleep message time-to-live is up. Nevertheless, wakeup messages can be sent from scripts as well if the application calls for it.

9.8 mask message

Mask messages, as the name suggests, are used to mask out incoming messages for the same resource ID as the mask message. A typical scenario is missing processes alarms as a result of running a cold database backup.

EMU processing:
call input script, add/update message, log message, call output script

Syntax:
$emsg1 [-h <hostname] [-u <user] -o mask -n <emu server -p <port -t <time-to-live -s <severity -w <password -c <class -m "<message"

Example:
a mask message from host dumbo.company.com.au that an oracle database called accounts is down. Since the backup should take a maximum of 3 hours, time-to-live is set to 3h. The mask message will be sent by an oracle cold backup script.
$emsg1 -o mask -n emuserver -p 2345 -t 3h-s 0 -c null -w icecream -m "accounts database is down"

Notice that the severity and class are set to 0 and null, respectively. It is because they are not used in mask messages.

Use:
If, for example, a process watcher is set up on a monitored system, there may be times that we want the process watcher disabled (masked out). It may be necessary if the process is missing for a reason, such as a cold backup.

9.9 query message

This message has two applications:
1. EMU database is queried whether a message for a queried resource ID exists in the database. emsg1 returns "none" if the message doesn't exist or the message type (normal,sleep, etc) if it does exist.

2. EMU is asked to produce a file from the out directory. emsg1 sends the file contents to its standard output. This option is great for agent configuration files distribution. On sending the file, EMU deletes it from the out directory.

EMU processing:
call input script, inquire on the message in the database or fetch a file in the out directory and send it to emsg1, log message, call output script

Syntax:
$emsg1 [-h <hostname>] [-u <user>] -o query -n <emu server> -p <port> -w <password> \
-m "<hostname>:<object ID>"

$emsg1 [-h <hostname>] [-u <user>] -o query -n <emu server> -p <port> -w <password> -m "FILE <file name>"

Example:
Script for job4 will send a query message to find out if a message about a completed job with resource ID = porky.company.com.au:job3 is in the EMU database.
If it is and it is of type "event", job 4 is run. Now it is obvious that job 4 starts off only if job 3 completed successfully.

```
RET=‘emsg1 -o query -n emuserver -p 2345 -w icecream -m " porky.company.com.au:job3"‘
if [ $? eq 1 ];then
      exit
fi
if [ "$RET" = "event" ];then
      run job 4
fi
```

emsg1 will return a "none" string if porky.company.com.au:job3 doesn’t exist in the database. Otherwise, it returns the type of message it found, e.g. "event" or "normal"

Another example is for agent configuration files distribution to a monitored host called dumbo. Configuration files are kept in one place on the management box. Each time an agent configuration file is changed, it is put in the EMU out directory (the out directory is specified in the <port.cfg EMU configuration file). The agent will check before each run if the EMU server has a new configuration file to download. If it doesn’t have any, the agent will use the old one. This scenario is good for sites without rcp capabilities that need central agent configuration file management. It is also good for nodes behind a firewall.

```
RET=‘emsg1 -o query -n emuserver -p 2345 -w icecream -m "FILE dumbo.cfg" | tee new_dumbo.cfg‘
if [ $? eq 1 ];then
   # a return code of 1 means that the connection to EMU timed out
   connect failed, try again later
   exit 1
fi
if [ "$RET" != "none" ];then
   cp new_dumbo.cfg dumbo.cfg
fi
```

run agents with dumbo.cfg

Use:
The above examples are quite descriptive to suggest the potential the command has. It can be used to facilitate dependency checks across multiple systems. The file to download can be a job to run, whereby EMU can be used as a simple scheduling system.

9.10 count <lag> message

Count messages are put in the text file only if more than the specified number of messages <lag value for the same resource ID are received in a row. It finds use in swap, CPU and memory monitoring.

EMU processing:
call input script, add/update with respect to the lag value, log message, call output script

Syntax:
$emsg1 [-h <hostname>] [-u <user>] -o "count <lag>" -n <emu server> -p <port> -t <time-to-live> -s <severity> \
-w <password> -c <class> -m "<message>"

Example:
A CPU agent on node porky scans CPU utilization every 5 minutes. In real life, it is not abnormal to see CPU utilization fluctuate for short periods of time. However, it is running at 100% for 25 minutes at a stretch, it may indicate a performance problem, possible due to a run-away process.

$emsg1 -o "count 5" -n emuserver -p 2345 -t 6m -s 1 -c /OS/UNIX/CPU -w icecream -m "CPU is 100% utilized"

Use:
Some resource we monitor may exhibit fluctuations that would normally distort a real problem detection. Introducing a lag allows to pick up cases of a long term continual resource problem.

9.11 event message

Event messages describe events. They just like normal messages with the difference that they are send once only on completion of an event. This event can be a batch job, backup etc. Time-to-live is set to how long we want EMU to know about this event for other scripts to query. Event messages don't show on the EMU browser.

EMU processing:
call input script, add/update message, log message, call output script

Syntax:
$emsg1 [-h <hostname>] [-u <user>] -o event -n <emu server> -p <port> -t <time-to-live> -s null -w <password> \
-c null -m "<object ID>"

Notice that the severity and class are set to 0 and null, respectively. It is because there is no use for them in event messages. The message text is just an object ID against which a query will be run.

-1- Example:
an event that job 4 completed so that other dependent jobs can run.
16$emsg1 -o event -n emu server -p 2345 -t 2h -s 0 -c null -w icecream -m "job"

Use:
The event message will find best use in synchronizing events across multiple systems.

9.12 suspend message

The suspend message suspends EMU (puts emu to sleep) for a specified number of seconds

EMU processing:
call input script, log message, sleep for a specified number of seconds

Syntax:
$emsg1 [-h <hostname>] [-u <user>] -o suspend -n <emu server> -p <port> -w <password> -m "<seconds>"

Example:
If we want to synchronize EMU on system A to EMU on system B, we need to perform a copy operation while system B is not undergoing any changes. By suspending system B for a short period of time, say 10 seconds, the database on system B can be copied to system A.

$emsg1 -o suspend -n emuserver -p 2345 -w icecream -m "10"

10 Scenarios of EMU use

1. hot backup script on speedy (application stays up)

sleep message <backup failed>
backup start
.
backup end
delete message <speedy:backup>

2. cold backup script on speedy (application1 is shut down)
sleep message <backup failed>
mask message <application1>
backup start
.

backup end
delete message <application>1
delete message <speedy:backup>

3. more to be added