Messaging for a Dynamic World

Mobeon Internal

| | |
|---|---|
| Approved: Per Berggren | No:1/BTS-MAS 0001 Uen |

| | | | |
|---|---|---|---|
| Copyright Mobeon AB<br>All rights reserved | Author: Kenneth Selin<br>Title: BTS - MAS | Version: C<br>Date: 2008-03-31 | 1/18 |

# BTS - MAS

## Content

History

| Version | Date | Adjustments |
|---|---|---|
| A | 2006-10-06 | First revision (ERMKESE) |
| B | 2007-06-13 | Minor changes. Description of realtime-thread related debugging (ermkese) |
| C | 2008-03-31 | Updated with SWA test cases. |

# 1 Introduction

This document describes basic test information for MAS.

**It is applicable if you are about to test an entire MAS.**

It documents the test tools needed, how to set up the environment for various tests, and the procedure for testing. It also specifies the test cases for basic test.

After reading this document, you will know how to set up, execute and document a basic test for this product.

**Note: if you choose to renumber the test cases in this document, you will need to renumber also the test cases in the BTR accordingly.**

# 2 Test Tools

## 2.1 Jhat

Jhat is a tool included in java 1.6. Jhat is used for checking memory leaks. Jhat reads a dump file of the java heap and starts a web server displaying the heap contents.

One advantage of using Jhat instead of Hat is that Jhat also supports OQL queries.

## 2.2 Hat

Hat is the same tool as Jhat (although older version), and was a tool outside of the Java SDK before Java 1.6.

## 2.3 Libcmem

Usage: libcmem <pid>

Shows calls to lmalloc, and lfree aggregated by the two topmost frame-pointers together with a calls to lmalloc/lfree from that site. Can be used to identify memory leaks by improper use of C-library routines.

Location: /vobs/ipms/mas/test/basictesttools

## 2.4 Tidprof

Works like dappprof from the DTraceToolkit, with the addition of a -t option so that you can single-out a specific thread by it's id (obtained through pstack,prstat -L etc). A typical call is tidprof -p 1234 -t 72 -U -u. For more information see DTraceToolkit documentation

Location: /vobs/ipms/mas/test/basictesttools

## 2.5 Malloctrace

Usage: malloctrace <pid>

Logs any failed malloc call with a stack trace pinpointing where it occurred.

Location: /vobs/ipms/mas/test/basictesttools

## 2.6 Dwhereami

Usage: dwhereami <-p pid | command>

Simple sampling profiler which is a good complement to dappprof/tidprof as it's not sensitive to functions that never returns (yes, those exists)

Location: /vobs/ipms/mas/test/basictesttools

## 2.7 Debug-start

Location: /vobs/ipms/mas/test/basictesttools

This is a script for starting MAS that makes a lot of assumptions. Just look at all these variables it takes for granted!

```
BASEDIR=`pwd`

BASE=$BASEDIR/MAS

LIBDIR=/vobs/ipms/mas/stream/lib

OPTIT=/usr/local/OptimizeitEntSuite2006

JDK=/usr/java/bin/java

LOG="/tmp/log"

CHANNELS=50
```

Fortunately we can change most of these with flags. Flags are given as 'flag' or 'flag=value' on the command line. No dash in the beginning is required or allowed.

- channels <n>
  Number of simultaneous channels, used to scale NewGen to a proper amount.   Should scale heap also, but doesn't. Use 'heap' flag to specify total heap.
- heap <n<M|G>>
  The size of the java heap, given in Megabyte or Gigabyte.
- log=<path>
  Path to logfiles, will only work for STDOUT and gc.log. 'mas.log' should be supported also, but needs additional work either in the Java code, or withxslt/xpath/sed/perl etc.
- jdk=<path to executable>
  Path to the Java executable, should really be called jvm. But nevertheless, it is still used to select which JDK to use.
- Opt
  Enables options suitable for OptimizeIT 2006. Uses additional files in VOB, all stored in mas/tools/optimezit. Uses default memory allocator.
- Debug
  Enables libumem with debug flags. The default backtrace is rather short, so it is not uncommon to be forced to add audit=45 to UMEM_DEBUG environment variable.
- Jconsole
  Enables jconsole.

## 2.8 Various standard test tools

This is a list of tools you may not use so often, but may find handy from time to time. They are standard, so you should be able to find their documentation.

- Dtrace. Useful but quite hard to use.
- Libumem. Useful but quite hard to use.
- sar
- mdb. Useful together with libumem.
- pmap
- pstack
- pargs
- gcore
- prtdiag
- jmap

- jconsole
- dbx
- gdb

# 3 Test Environment

## 3.1 Memory tests

To verify that all objects can be garbage collected, you can test this as follows:

Set up MAS with a big log area, allow for example 100 log files of 100 MB each.

Start the JVM with flag -XX:+HeapDumpOnOutOfMemoryError, which causes the JVM to make a “jmap style” dump file in case the heap gets full, which can then be analysed with jhat.

To have the OutOfMemoryException generated faster, decrease the heap size to e.g. 50M, by using these flags to the JVM: -Xms50m –Xmx50m

### 3.1.1 Using hat

Hat is started as follows:

```
hat -jar hat.jar -stack false java_pid25508.hprof
```

where ”heap.bin” is the assumed file name of the file created by the JVM at OutOfMemoryException.

### 3.1.2 Using jhat

Jhat is started as follows:

```
jhat java_pid25508.hprof
```

This is an OQL query to check what CCXMLEventHub objects that are still on the heap:

```
select {id:h.executionContext.session.id.id,context:h} from
com.mobeon.masp.execution_engine.ccxml.runtime.CCXMLEventHub h where
h.executionContext.session.id.id >= 0 &&
!contains(referrers(h),"identical(classof(it),heap.findClass('com.mobeon.masp.exe
cution_engine.runtime.event.EventHubImpl$2'))") &&
count(filter(unique(h.executionContext.eventSourceManager.connectionsById.tabl
e), "it != null"))>0 &&
filter(unique(h.executionContext.eventSourceManager.connectionsById.table), "it
!= null")[0].value.state.value.name.value.toString() == "DISCONNECTED"
```

# 4 Test Execution

This paragraph is intentionally left blank.

# 5 Test Cases

## 5.1 Installation

This section is used for test installation on Solaris 10.

### 5.1.1 Clean installation without responsefile.

Verify that installation without responsefile can be done.

Action.

- Install latest Emanate version prior to install MAS.
- Follow the installation guide to install MAS with no responsefile.

Result

- Look in the install log file to verify that no errors have occurred.
- Verify that emanate path is correct by sending unlock/viewmib to MAS (rc.mas unlock, rc.mas viewmib)

### 5.1.2 Clean installation with responsefile.

Verify that installation with a responsefile can be done.

Action.

- Install latest Emanate version prior to install MAS.
- Follow the installation guide to install MAS with a responsefile.

Result

- Look in the install log file to verify that no errors have occurred.
- Verify that emanate path is correct by sending unlock/viewmib to MAS (rc.mas unlock, rc.mas viewmib)

### 5.1.3 Clean installation with faulty responsefile.

Verify that installation with a faulty responsefile can’t be done.

Action.

- Install latest Emanate version prior to install MAS.

- After created the responsefile, edit the responsefile and remove one parameter from the responsefile.

- Follow the installation guide to install MAS with a responsefile.

Result

- Verify that the installation is aborted.

#### 5.1.4 Clean installation with old emanate(no package) version.

Verify that MAS can handle old Emanate versions .

Action

- Install an old emanate version. Emanate P12A.010 (no package installation). Follow the installation guide to install MAS.
- Look in the install log file to verify that no errors have occurred.
- Verify that emanate path is correct by sending unlock/viewmib to MAS (rc.mas unlock, rc.mas viewmib)

Result

- Verify that no errors occurs doing unlock/viewmib to MAS.

#### 5.1.5 Clean installation with old emanate(package, no BIN_DIR) version.

Verify that MAS can handle old Emanate versions .

Action

- Install an old emanate version. Emanate P12B.002 (package, but no SNMP_BIN_DIR set) installation). Follow the installation guide to install MAS.
- Look in the install log file to verify that no errors have occurred.
- Verify that emanate path is correct by sending unlock/viewmib to MAS (rc.mas unlock, rc.mas viewmib)

Result

- Verify that no errors occurs doing unlock/viewmib to MAS.

### 5.2 Various tests

#### 5.2.1 Stop

Verify that MAS handle the kill trap and that nothing unexpected is written to the process.log after the stop.

Action

- Install MAS according to the installation guide.

- Start MAS and verify in the mas.log file that MAS is started.
- Use rc.mas stop to stop MAS.

Result

- The last entry in the mas.log file should be "MAS Shutdown complete."
- After the stop, nothing unexpected should be written to process.log

### 5.2.2 Wrapping process.log

Verify that process.log store the current log file with a .1 extension.

Action

- Install MAS according to the installation guide.
- Update the MAX_FILESIZE=5000000 to 100. This is only to force process .log to wrap when file size reaches 100kb.
- Start MAS and verify in the mas.log file that MAS is started.
- Make MAS produce stdout output.
  Update logmanager.xml file. In the root group change the **<appender-ref ref="ROLLFILE" />** to **<appender-ref ref="STDOUT" />** and **<priority value ="warn" />** to **<priority value ="debug" />**.
- Generate output by sending rc commands. I.e. lock, unlock, viewmib, reloadconfig.
- Look in the log directory and verify that process.log never should be larger then 100kb and that a process.log.1 is created.

Result

- Verify that when process.log reaches 100kb, it is renamed to process.log.1, and logging continues in a newly created process.log.

### 5.2.3 MCR register with no parameters

Verify that when registering to MCR. Appropriate actions are made when registering.

If error registering due to wrong parameters, a cron entry should not be added.
If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrreg.sh script with no parameters
  *#./mcrreg.sh*

Result

- Verify that "Parameter Error. No retries will be attempted" will be printed

### 5.2.4 MCR register with faulty parameters 1

Verify that when registering to MCR. Appropriate actions are made when registering.

If error registering due to wrong parameters, a cron entry should not be added. If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrreg.sh script with testservice as parameter
  *#./mcrreg.sh testservice*

Result

- Verify that "Parameter Error. No retries will be attempted" will be printed

### 5.2.5 MCR register with faulty parameters 2

Verify that when registering to MCR. Appropriate actions are made when registering.

If error registering due to wrong parameters, a cron entry should not be added. If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrreg.sh script with "register testservice unknownprop=unknown" as parameters
  *#./mcrreg.sh testservice  unknownprop=unknown*

Result

- Verify that "Parameter Error. No retries will be attempted" will be printed

### 5.2.6 MCR register with correct parameters

Verify that when registering to MCR. Appropriate actions are made when registering.

If error registering due to wrong parameters, a cron entry should not be added. If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrreg.sh script with the following parameters :
  testservice

component=testcomp
type=testtype
hostname=hostname
version=version
*#./mcrreg.sh testservice component=testcomp type=testtype hostname=hostname version=version*

- Verify that "Registration ok" will be printed
- Verify that no cron entries are added to cron.

### 5.2.7 MCR register with correct parameters but MCR unreachable

Verify that when registering to MCR. Appropriate actions are made when registering.

If error registering due to wrong parameters, a cron entry should not be added.
If error registering due to MCR not reachable, a cron entry should be added.

Action

- On an installed MAS, update the mas.xml file.
  In the group <externalcomponentregister>. Locate hostname="mcrhost" entry and change mcrhost to a host with no MCR. I.e. to hostname="localhost"
- Run the mas/etc/mcrreg.sh script with the following parameters :
  testservice
  component=testcomp
  type=testtype
  hostname=hostname
  version=version
  *#./mcrreg.sh testservice component=testcomp type=testtype hostname=hostname version=version*
- Verify that "MCR unreachable. MCR registration will be retried later" will be printed.
- Verify that a cron entry is added to cron and that the entry is correct.
  *#crontab –l*
  *#.....*
  *#0,5,10,15,20,25,30,35,40,45,50,55 * * * * /apps/mas/etc/mcrlib register testservice component=testcomp type=testtype hostname=hostname version=version logicalzone=ipms.lab.mobeon.com hostname=bilbo.ipms.lab.mobeon.com > /dev/null 2>&1*

### 5.2.8 MCR unregister with no parameters

Verify that when registering to MCR. Appropriate actions are made when registering.

If error registering due to wrong parameters, a cron entry should not be added.
If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrunreg.sh script with no parameters
  *#./mcrunreg.sh*

Result

- Verify that "Parameter Error. No retries will be attempted" will be printed

### 5.2.9 MCR unregister with faulty parameters 1

If error registering due to wrong parameters, a cron entry should not be added.
If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrunreg.sh script with testservice as parameter
  *#./mcrunreg.sh testservice*

Result

- Verify that "Parameter Error. No retries will be attempted" will be printed

### 5.2.10 MCR unregister with correct parameters

If error registering due to wrong parameters, a cron entry should not be added.
If error registering due to MCR not reachable, a cron entry should be added.

Action

- Make sure that MCR is reachable.
- On an installed MAS, run the mas/etc/mcrunreg.sh script with the following parameters :
  testservice
  component=testcomp
  *#./mcrunreg.sh testservice component=testcomp*

Result

- Verify that "Unregistration ok" will be printed.

### 5.2.11 MCR unregister with correct parameters but MCR unreachable

If error registering due to wrong parameters, a cron entry should not be added.
If error registering due to MCR not reachable, a cron entry should be added.

Action

- On an installed MAS, update the mas.xml file.
  In the group <externalcomponentregister>. Locate hostname="mcrhost" entry and change mcrhost to a host with no MCR. I.e. to hostname="localhost"
- Run the mas/etc/mcrunreg.sh script with the following parameters : testservice component=testcomp
  *#./mcrreg.sh testservice component=testcomp*
- Verify that “MCR unreachable. No retries will be attempted” will be printed.
- Verify that a cron entry is not added to cron.
  #crontab -l

## 5.3 System Wide Announcement tests

### 5.3.1 Install SWA.

Verify that a SWA delivery can be installed.

Action

- Use the swaadmin.sh script to install the delivery. Run the command:
  `/apps/mas/bin/swaadmin.sh install <mcpid> <swa_file>`

Result

- Verify that the new prompts are copied to correct location. Call in and verify that the SWA is played.

### 5.3.2 Uninstall SWA.

Verify that a SWA delivery can be uninstalled.

Action

- Use the swaadmin.sh script to uninstall the delivery. Run the command:
  `/apps/mas/bin/swaadmin.sh uninstall <mcpid>`

Result

- Verify that the new prompts are removed and the old (silent) prompts should have been put back. Call in and verify that no SWA is played.

# 6 Reporting Test Results

This paragraph is intentionally left blank.

# 7 References

**[1]** <Document name>
<Document number>

# 8 Terminology

| Term | Explanation |
|---|---|

# 9 Appendix A: debugging a hung Unix host

For example when using real time threads encountering spin locks, it is possible that the entire Unix host hangs. Below is a procedure how you may debug such a case.

First, make sure that dumping the physical memory is enabled:

dumpadm -c all -s /apps/crash/masX

Now, when the host hangs, you issue the command “sync” from the console, which will dump the memory to the directory specified above.

When the host is rebooted, you analyse the dumped files as follows.

$ mdb -k unix.0 vmcore.0

Loading modules: [ unix krtld genunix specfs dtrace ufs sd md ip sctp usba fctl random lofs ptm cpc fcip crypto nfs ]

# List all real time threads in the state TS_ONPROC

::sizeof sclass_t | >p2;::walk cpu p1 |::print cpu_t cpu_dispthread |::print -a kthread_t t_cid | ::map sclass+(<p2*.) | ::print sclass_t cl_init | ::grep .==rt_init |::eval <p1=K | ::print cpu_t cpu_dispthread

..thread list..

# Verify process belonging.

::sizeof sclass_t | >p2;::walk cpu p1 |::print cpu_t cpu_dispthread |::print -a kthread_t t_cid | ::map sclass+(<p2*.) | ::print sclass_t cl_init | ::grep .==rt_init |::eval <p1=K | ::print cpu_t cpu_dispthread | ::print kthread_t t_procp| ::print proc_t p_user.u_psargs

p_user.u_psargs = [ "/apps/jdk1.5.0_11/bin/java -server -Xms207M -Xmx207M -XX:+ParallelRefProcEnable" ] p_user.u_psargs = [ "/apps/jdk1.5.0_11/bin/java -server -Xms207M -Xmx207M -XX:+ParallelRefProcEnable" ]

# All real time threads in this case belong to a java process with the same arguments.

```
# ... Alternatively, show pid instead of process arguments process arguments

::sizeof sclass_t | >p2;::walk cpu p1 |::print cpu_t cpu_dispthread |::print -a
kthread_t t_cid | ::map sclass+(<p2*.) | ::print sclass_t cl_init | ::grep
.==rt_init |::eval <p1=K | ::print cpu_t cpu_dispthread | ::print kthread_t
t_procp| ::print "struct proc" p_pidp | ::print "struct pid" pid_id |="pid="D"\n"


pid=10850

pid=10850

# All real time threads in this case belong to pid 10850

# Select one of the java threads and run corresponding to:

> 30002b552c0::threadlist

            ADDR             PROC              LWP CMD/LWPID

0000030002b552c0      30000328010      30005d96a98 java/128



# PROC is our processadress, it is used later to debug the java process.

# Print the actual frame pointers and pc for all our RT threads.


 ::sizeof sclass_t | >p2;::walk cpu p1 |::print cpu_t cpu_dispthread  |::print -a
kthread_t t_cid | ::map sclass+(<p2*.) | ::print sclass_t  cl_init | ::grep
.==rt_init |::eval <p1=K | ::print cpu_t  cpu_dispthread |::print kthread_t t_lwp-
>lwp_regs | ::print "struct regs" r_o6 r_pc


r_o6 = 0xe007b438

r_pc = 0xfefa9258


# In this case there was just one thread, but if there are several threads, the
printout is repeated once per thread.


# Now we leave kernel mode to debug the java process.

# This is since all addresses are in the process' address space but in a kernel
dump, mdb works in the address space of the kernel. Use the PROC address you
extracted earlier.


> 30000328010::context

debugger context set to proc 30000328010
```

```
# If you get the message "mdb: dump does not contain pages for # proc
30000328010" you must make sure to run the ”dumpadm” command above and
then wait for a new, correct dump.

# Start with checking where the first thread’s r_pc points:

> 0xfefa9258/

0xfefa9258:     80a6e000        = cmp       %i3, 0


# There was no symbol information there. We select the next frame.


> 0xe007b438/56+"fp "X" pc "X

0xe007b438:     fp e007b4a8         pc fefaadf8


# ... and dump r_pc


> fefaadf8/

liblog4cxx.so`_ZN7log4cxx7helpers6Thread20InterlockedIncrementEPVl+8:
                7ffff910        = call      -0x1bc0       <0xfefa9238>


# Now we found symbol information showing we are in
ZN7log4cxx7helpers6Thread20InterlockedIncrementEPVl

# If we want it more readable, we can use:

> dem ZN7log4cxx7helpers6Thread20InterlockedIncrementEPVl

_ZN7log4cxx7helpers6Thread20InterlockedIncrementEPVl ==

log4cxx::helpers::Thread::InterlockedIncrement


# This is the second stack level. Following levels follow the same simple pattern.

# Take fp from previous stack, print next fp and pc. Dump the pc.

> e007b4a8/56+"fp "X" pc "X

0xe007b4a8:     fp e007b518         pc fef6fcec


> fef6fcec/

liblog4cxx.so`_ZN7log4cxx9Hierarchy9getLoggerERKSs+0x7c:        9fc30000
                = jmpl      %o4, %o7
```

## 9.1 Dump fields on the stack

If you want to dump stack fields for example to check function arguments, follow the procedure below.

```
> 0xe007b4a8::vtop

virtual e007b4a8 mapped to physical 20935b4a8

> 0::context

debugger context set to kernel

> 20935b4a8::print -p "struct frame32"

{

   fr_local = [ 0xff38d108, 0x39b68, 0x39b50, 0x39b80, 0x178, 0x36000, 0,
0xfee7faa0 ]

   fr_arg = [ 0x6dcf94, 0xff2b8f39, 0xe007b6d8, 0xe007b56f, 0xfefd9fd4, 0x8 ]

   fr_savfp = 0xe007b518

   fr_savpc = 0xfef6fcec

   fr_stret = 0x194e318

   fr_argd = [ 0x21, 0xe007b548, 0xf8805764, 0x1, 0x682, 0xe5c97400 ]

   fr_argx = [ 0xff16cbc0 ]

}

> 30000328010::context

debugger context set to proc 30000328010


# But you would have to switch between kernel and process mode all the time.

# If you want other fields of the stack for other levels, you can do as follows:

# First switch to kernel mode:

 >0::context

# Print the layout of a 32-bit frame:

> ::print -t "struct frame32"

{

   int [8] fr_local

   int [6] fr_arg

   caddr32_t fr_savfp

   int fr_savpc

   caddr32_t fr_stret

   int [6] fr_argd
   int [1] fr_argx
}
>
```

```
# If we now would like to print fr_argx, we first find its offset:
> ::offsetof "struct frame32" fr_argx
offsetof (struct frame32, fr_argx) = 0x5c
```

```
# Calculate the correct offset relatively all fields we print lying before in the
structure, and for all offsets of the form (nn+) lying there:
 > 0x5c-0t56-0t4-0t4=d
28
```

```
# 0t means the in-data is decimal, =d that we want to have the answer in
decimal.
# 0x5c = offset
# 0t56 = offset of sav_fp
# 0t4 = read 4 bytes (fp in X format, that is hex 32 bits )
# 0t4 = read 4 bytes (pc in X format, that is hex 32 bits )
```

```
# Then we add it to the end of the expression we have to print a frame. We also
add a few \n for cuter printouts:
# /56+"fp "X" pc "X instead becomes
# /56+"fp "X"\npc "X"\nargx "32+X, and hence
> e007b4a8/56+"fp "X"\npc "X"\nargx "28+X
0xe007b4a8:     fp e007b518
                pc fef6fcec
                argx ff16cbc0
```

```
# To verify this, we can use
> 0xe007b4a8::vtop
virtual e007b4a8 mapped to physical 20935b4a8
> 0::context
debugger context set to kernel
> 20935b4a8::print -p "struct frame32"
```

```
{
   fr_local = [ 0xff38d108, 0x39b68, 0x39b50, 0x39b80, 0x178, 0x36000, 0,
0xfee7faa0 ]
   fr_arg = [ 0x6dcf94, 0xff2b8f39, 0xe007b6d8, 0xe007b56f, 0xfefd9fd4, 0x8 ]
   fr_savfp = 0xe007b518
   fr_savpc = 0xfef6fcec
   fr_stret = 0x194e318
   fr_argd = [ 0x21, 0xe007b548, 0xf8805764, 0x1, 0x682, 0xe5c97400 ]
   fr_argx = [ 0xff16cbc0 ]
}
```