

MEMORYSCAPE

**VARIABLES AND
OPTIONS**



VERSION 2.4.0



Copyright © 2007–2008 by TotalView Technologies, LLC.
Copyright © 1999–2008 by Etnus LLC. All rights reserved.
Copyright © 1998–1999 by Etnus, Inc.
Copyright © 1996–1998 by Dolphin Interconnect Solutions, Inc.
Copyright © 1993–1996 by BBN Systems and Technologies, a division of BBN Corporation.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the prior written permission of TotalView Technologies LLC. (TotalView Technologies).

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

TotalView Technologies has prepared this manual for the exclusive use of its customers, personnel, and licensees. The information in this manual is subject to change without notice, and should not be construed as a commitment by TotalView Technologies. TotalView Technologies assumes no responsibility for any errors that appear in this document.

TotalView and TotalView Technologies are registered trademarks of TotalView Technologies LLC.

TotalView uses a modified version of the Microline widget library. Under the terms of its license, you are entitled to use these modifications. The source code is available at <http://www.totalviewtech.com/Products/TotalView/developers>.

All other brand names are the trademarks of their respective holders.

Contents

1 The TVHEAP_ARGS Environment Variable

Block Painting Keywords	1
Event Reporting Keywords	2
Guard Blocks Keywords	4
Hoarding Keywords	5
Logging Keywords	5
Other Keywords	6

2 tvdsvr Command and Options

Format	7
Description	7
Options.....	7

3 tvdsvr Replacement Characters

Replacement Character Reference	9
---------------------------------------	---

Chapter 1 **The TVHEAP_ARGS Environment Variable**

When debugging multi-process, multi-threaded programs, MemoryScape pushes the configuration options you set into the processes and threads your program creates. In some cases, it cannot do this. For example, if you start an MPI program using either `mpirun` or `mpiexec` commands. Another example is that there may be difficulties setting these options when your program is being started using a batch system.

This appendix describes the `TVHEAP_ARGS` environment variable. You will enter keywords that identify MemoryScape options. Associated with each keyword is the value to which the option will be set.

As the MemoryScape agent starts up, it checks to see if this variable is set. If it is, it sets its options based on what you have entered. Here's a C Shell example:

```
setenv TVHEAP_ARGS="paint_on_alloc=true  
                    notify_alloc_null=true"
```

Each option/argument pair is separated from another option/argument pair by a space character. (Due to printing limitations, this isn't visible in the print version of this text.

As an alternative, you can enter these options into a file. When entered into a file, place each option/value pair on its own line. For example:

```
paint_on_alloc=true  
notify_alloc_null=true
```

The file containing these pairs should have a `.hiarc` extension. If you name the file `default.hiarc`, you can place it in the following three places:

- In a subdirectory within your home directory named `.totalview/hia`.
- In your current working directory; that is, the directory from which you invoked MemoryScape.
- In the directory containing the executable that you will be debugging.

If you wish to place this file elsewhere or give it a different name, you will need to set the `TVHEAP_CONFIG_FILE` environment variable. For example:

```
setenv TVHEAP_CONFIG_FILE "/home/me/stuff.hiarc"
```

Block Painting Keywords

When your program allocates or deallocates a block, MemoryScape can paint the block with a bit pattern. This makes it easy to identify uninitialized blocks, or blocks pointed to by dangling pointers.

Some heap allocation routines such as `calloc()` return memory initialized to zero. Using the `paint_on_zalloc` option allows you to separately enable the painting of the memory blocks altered by these kinds of routines. If you do enable painting for routines that set memory to zero, MemoryScape uses the same pattern that it uses for a normal allocation.

Block painting keywords are:

paint_on_alloc Setting this option to **true** tells MemoryScape to paint a memory block when your program's heap manager allocates it. The pattern used is either the default of the pattern used set using the `paint_alloc_pattern`.

Setting this to **false** disables block painting for allocations.

default: **false**

paint_on_dealloc

Setting this option to **true** tells MemoryScape to paint a memory block when your program's heap manager deallocates it. The pattern used is either the default of the pattern used set using the `paint_dealloc_pattern`.

Setting this to **false** disables block painting for deallocations.

default: **false**

paint_on_zalloc Setting this option to **true** tells MemoryScape to paint a memory block when your program's heap manager used a routine such as `calloc()` that sets the memory pattern to all zeros. The pattern used is ei-

ther the default of the pattern used set using the `paint_alloc_pattern`.

You can only paint zero-allocated blocks if you are also painting regular allocations.

Setting this to **false** disables block painting for zero-allocated blocks.

This option can crash your program as most programs that zero-fill memory rely on those values.

default: **false**

paint_alloc_pattern

Set the pattern that MemoryScape uses when it paints a block of memory immediately after it is allocated. The maximum width of *pattern* is eight bytes. However, your pattern can be shorter.

default: **0xa110ca7f**

paint_dealloc_pattern

Set the pattern that MemoryScape uses when it paints a block of memory immediately after it is allocated. The maximum width of *pattern* is eight bytes. However, your pattern can be shorter.

default: **0xdea110cf**

Event Reporting Keywords

The keywords described in this section tell MemoryScape if it should stop execution when an event occurs. They also let you individually specify which events will stop execution.

Event reporting keywords are:

tv_stop_on_event

When set to **true**, MemoryScape stops execution when an event occurs. When set to **false**, MemoryScape ignores the event. Stopping event notification does not change the way in which the

MemoryScape records memory allocations, reallocations, and deallocations. It just tells MemoryScape that execution shouldn't stop.

default: **true**

enable_event_filtering

When set to **true**, you must can individually use the notify options to control which events stop execution. When set to **false**, any event that MemoryScape detects will stop execution.

default: **true**

notify_free_not_allocated

When set to **true**, MemoryScape stops execution when your program attempts to free a block.that is not allocated. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_realloc_not_allocated

When set to **true**, MemoryScape stops execution when your program attempts to reallocate a block.that is not allocated. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_addr_not_at_start

When set to **true**, MemoryScape stops execution when your program attempts to free a block using an address that points into the interior of an allocated block rather than to the address returned when the block was allocated. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_double_alloc

When set to **true**, MemoryScape stops execution when a malloc library routine attempts to allocate memory because it returned the address of a block already allocated. This indicates a problem in the malloc library, not in the application. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_alloc_not_in_heap

When set to **true**, MemoryScape stops execution when your program attempts to deallocate memory and MemoryScape cannot locate the memory block within it heap tables. When set to **false**, MemoryScape ignores this event.

default: **false**

notify_alloc_null

When set to **true**, MemoryScape stops execution when your program attempts to allocate memory and no more memory is available. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_alloc_returned_bad_alignment

When set to **true**, MemoryScape stops execution when your program attempts to allocate a memory block and the address returned is not aligned correctly. This can indicate a problem with your system's memory allocator. However, it can also occur if your program overwrites data structures used by the malloc library (This error is very rare.) When set to **false**, MemoryScape ignores this event.

default: **true**

notify_bad_alignment_argument

When set to **true**, MemoryScape stops execution when your program attempts to allocate memory using a call such as **memalign()** and the program's argument would force an alignment problem. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_double_dealloc

When set to **true**, MemoryScape stops execution when your program attempts to free a block.that was already freed. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_guard_corruption

When set to **true**, MemoryScape stops execution when your program frees a block that had guard blocks and whose guard blocks it detects that a guard block was altered. This alteration indicates that your program wrote data where it should not have. When set to **false**, MemoryScape ignores this event.

default: **true**

notify_termination

When set to **true**, MemoryScape stops execution when your program is about to exit. When set to **false**, MemoryScape does not stop execution at this time.

The exit event is raised when the agent library is being cleaned up, which is after the call to the **exit()** function.

default: **true**

notify_dealloc

When set to **true**, MemoryScape stops execution when your program deallocates a watched memory block. When set to **false**, MemoryScape does not stop execution at this time.

Blocks are placed on this list by selecting a check box within the Block Properties window.

default: **true**

notify_realloc

When set to **true**, MemoryScape stops execution when your program reallocates a watched memory block. When set to **false**, MemoryScape does not stop execution at this time.

Blocks are placed on this list by selecting a check box within the Block Properties window.

default: **true**

Guard Blocks Keywords

When your program allocates a memory block, MemoryScape can surround this block with additional memory. It will also initialize this memory

to a bit pattern. When the MemoryScape checks these blocks, it can tell if your program overwrote the blocks.

If you are using the GUI, you would have set these options using the following controls:

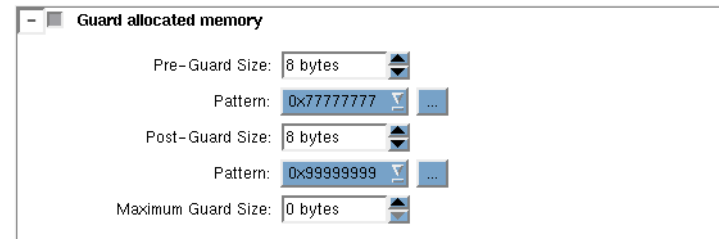


Figure 1:

Block painting keywords are:

enable_guard_blocks

When set to **true**, MemoryScape write guard blocks around memory blocks allocated on the heap. When set to **false**, it does not write guard blocks.

default: **false**

guard_max_size Specifies the maximum size for the guard blocks that surround one memory allocation. If memory is tight, setting a value here can limit the amount of blocks that are used. If this size is exceeded, MemoryScape doesn't create blocks for the allocation.

The size that is actually used can be greater than the pre-guard and post-guard sizes due to the way an operating system aligns information. If set to zero (0), which is the default, MemoryScape doesn't set a maximum size.

default: **0**

guard_pre_size Specify the size in bytes of the guard block that precedes a memory allocation. MemoryScape may use a different size if it needs to adjust the size to meet alignment and allocation unit size constraints. If the guard size will be greater than this maximum guard size, the Memory Debugger does not create a guard.

default: 8

guard_post_size Specify the size in bytes of the guard block that follows a memory allocation. MemoryScape may use a different size if it needs to adjust the size to meet alignment and allocation unit size constraints. If the guard size will be greater than this maximum guard size, the Memory Debugger does not create a guard.

default: 8

guard_pre_pattern Defines the pattern that MemoryScape uses when it writes a guard block that precedes a memory allocation. If you omit this keyword, MemoryScape uses its default pre-guard pattern.

default: 0x77777777

guard_post_pattern Defines the pattern that MemoryScape uses when it writes a guard block that follows a memory allocation. If you omit this keyword, MemoryScape uses its default pre-guard pattern.

default: 0x99999999

Hoarding Keywords

In some cases, you may not want your system's heap manager to immediately reuse memory. You would do this, for example, when you are trying to find problems that occur when more than one process or thread is allocating the same memory block. Hoarding allows you to temporarily delay the block's release to the heap manager. When the hoard has reached its capacity in either size or number of blocks, the Memory

Debugger releases previously hoarded blocks back to your program's heap manager.

The order in which the Memory Debugger releases blocks is the order in which it hoards them. That is, the first blocks hoarded are the first blocks released—this is a first-in, first-out (fifo) queue.

As your program executes, the Memory Debugger adds the deallocated region to a FIFO buffer. Depending on your program's use of the heap, the hoard could become quite large. You can control the hoard's size by setting the maximum amount of memory in kilobytes that the Memory Debugger can hoard and the maximum number of hoarded blocks.

If you are using the GUI, you would have set these options using the following controls:

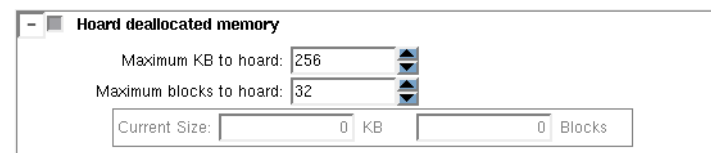


Figure 2:

Hoarding keywords are:

enable_hoarding When set to true, MemoryScape does not to surrender allocated blocks back to your program's heap manager.

default: false

hoard_maximum_num_blocks Set the maximum number of hoarded blocks.

default: 32

hoard_maximum_kb Sets the maximum size of the hoarded information.

default: 256

Logging Keywords

MemoryScape can write information about the debugging session as they occur to a logging file. In addition, it can write information about memory blocks that were not deallocated when your program exits into this same logging file.

Logging keywords are:

output_fd	Names the file descriptor that MemoryScape uses when it writes debugging data. Use this keyword if you cannot not use an output_file keyword.
<i>default:</i> 1	
output_file	Names the file into which MemoryScape writes debugging data.
<i>default:</i> heap.out	
verbosity	Sets the MemoryScape verbosity level. If the level is greater than 0, MemoryScape sends information to stderr . The values you can set are:
	0: Display no information. This is the default.
	1: Print error messages.
	2: Print all relevant information.
<i>default:</i> 0	
display_allocations_on_exit	When set to true , MemoryScape writes information on allocations to the output file just before your program exits.
<i>default:</i> true	

Other Keywords

This section describes keywords that do not fit into any of the previous categories. In addition they are seldom used. For example, you would not

alter the backtrace depth unless you find that running your program with MemoryScape means that you run out of memory. These other keywords are:

backtrace_depth	Tells the Memory Debugger to limit the number of stack frames to the value that you type as an argument. The <i>depth</i> is the maximum number of PCs that the Memory Debugger includes when it creates a backtrace. (The backtrace is created when your program allocates or reallocates a memory block.) The <i>depth</i> value starts after the <i>trim</i> value. That is, the number of excluded frames does not include the trimmed frames.
<i>default:</i> 32	
backtrace_trim	Tells MemoryScape to remove—that is, trim—this number of stack frames from the top of the backtrace. The <i>trim</i> describes the number of PCs from the top of the stack that MemoryScape ignores when it creates a backtrace. As the backtrace includes procedure calls from within MemoryScape, setting a trim value removes them from the backtrace. The default is to exclude MemoryScape procedures. Similarly, your program might call the heap manager from within library code. If you do not want to see call frames showing a library, you can exclude them.
<i>default:</i> -1	
memalign_strict_alignment_even_multiple	MemoryScape provides an integral multiple of the alignment rather than the even multiple described in the Sun memalign documentation. By including this value, you are telling MemoryScape to use the Sun alignment definition. However, your results might be inconsistent if you do this.
<i>default:</i> false	

Chapter 2

tvdsvr Command and Options

If you are altering a startup preference, you will also need the information located within “tvdsvr Replacement Characters” on page 9.

Format

```
tvdsvr {-server | -callback hostname:port | -serial device}
      [other options]
```

Description

The **tvdsvr** debugger server allows MemoryScape to control and debug a program on a remote machine. To accomplish this, the **tvdsvr** program must run on the remote machine, and it must have access to the executables being debugged. These executables must have the same absolute path name as the executable that MemoryScape is debugging, or the **PATH** environment variable for **tvdsvr** must include the directories containing the executables.

You must specify a **-server**, **-callback**, or **-serial** option with the **tvdsvr** command. By default, MemoryScape automatically launches **tvdsvr** using the **-callback** option, and the server establishes a connection with

MemoryScape. (Automatically launching the server is called *autolaunching*.)

If you prefer not to automatically launch the server, you can start **tvdsvr** manually and specify the **-server** option. Be sure to note the password that **tvdsvr** prints out with the message:

```
pw = hexnumhigh:hexnumlow
```

MemoryScape will prompt you for *hexnumhigh:hexnumlow* later. By default, **tvdsvr** automatically generates a password that it uses when establishing connections. If desired, you can set your own password by using the **-set_pw** option.

Options

The following options name the port numbers and passwords that MemoryScape uses to connect with **tvdsvr**.

```
-callback hostname:port
```

(Autolaunch feature only) Immediately establishes a connection with a MemoryScape process running on *hostname* and listening on *port*, where *hostname* is either a host name or TCP/IP address. If **tvdsvr** cannot connect with MemoryScape, it exits.

Options

If you use the `-port`, `-search_port`, or `-server` options with this option, `tvdsvr` ignores them.

`-callback_host` *hostname*

Names the host upon which the callback is made. The *hostname* argument indicates the machine upon which MemoryScape is running.

`-callback_ports` *port-list*

Names the ports on the host machines that are used for callbacks. The *port-list* argument contains a comma-separated list of the host names and TCP/IP port numbers (*hostname:port,hostname:port...*) on which MemoryScape is listening for connections from `tvdsvr`. This option is most often used with a bulk launch.

`-debug_file` *console_outputfile*

Redirects TotalView Debugger Server console output to a file named *console_outputfile*.

default: All console output is written to `stderr`.

`-port` *number*

Sets the TCP/IP port number on which `tvdsvr` should communicate with MemoryScape. If this port is busy, `tvdsvr` does not select an alternate port number (that is, it won't communicate with anything) unless you also specify `-search_port`.

default: 4142

`-search_port`

Searches for an available TCP/IP port number, beginning with the default port (4142) or the port set with the `-port` option and continuing until one is found. When the port number is set, `tvdsvr` displays the chosen port number with the following message:

```
port = number
```

Be sure that you remember this port number, since you will need it when you are connecting to this server from MemoryScape.

`-server`

Listens for and accepts network connections on port 4142 (default).

Using `-server` can be a security problem. Consequently, you must explicitly enable this feature by placing an empty file named `tvdsvr.conf` in your `/etc` directory. This file must be owned by user ID 0 (root). When `tvdsvr` encounters this option, it checks if this file exists. This file's contents are ignored.

You can use a different port by using one of the following options: `-search_port` or `-port`. To stop `tvdsvr` from listening and accepting network connections, you must terminate it by pressing Ctrl+C in the terminal window from which it was started or by using the `kill` command.

`-set_pw` *hexnumhigh:hexnumlow*

Sets the password to the 64-bit number specified by the *hexnumhigh* and *hexnumlow* 32-bit numbers. When a connection is established between `tvdsvr` and MemoryScape, the 64-bit password passed by MemoryScape must match this password set with this option. `tvdsvr` displays the selected number in the following message:

```
pw = hexnumhigh:hexnumlow
```

We recommend using this option to avoid connections by other users.

If necessary, you can disable password checking by specifying the `"-set_pw 0:0"` option with the `tvdsvr` command. Disabling password checking is dangerous; it allows anyone to connect to your server and start programs, including shell commands, using your UID. Therefore, we do not recommend disabling password checking.

`-verbosity` *level*

Sets the verbosity level of `tvdsvr`-generated messages to *level*, which may be one of `silent`, `error`, `warning`, or `info`.

default: `info`

`-working_directory` *directory*

Makes *directory* the directory to which MemoryScape will be connected.

Note that the command assumes that the host machine and the target machine mount identical file systems. That is, the path name of the directory to

which MemoryScape is connected must be identical on both the host and target machines.

After performing this operation, the **tvdsvr** is started.

Chapter 3

tvdsvr Replacement Characters

If you are altering a startup preference, you will also need the information located within “*tvdsvr Command and Options*” on page 7.

When placing a **tvdsvr** command in a **Server Launch** string, you will need to use special replacement characters. When your program needs to launch a remote process, MemoryScape replaces these command characters with what they represent.

Replacement Character Reference

Here are the replacement characters:

%B	Expands to the bin directory where tvdsvr is installed.
%C	Is replaced by the name of the server launch command being used. On most platforms, this is rsh . On HP, this command is remsh . If the TVDSVRLAUNCH-CMD environment variable exists, MemoryScape will use its value instead of its platform-specific value.
%D	Is replaced by the absolute path name of the directory to which MemoryScape will be connected.
%F	Contains the “tracer configuration flags” that need to be sent to tvdsvr processes. These are system-specific startup options that the tvdsvr process needs.

%I	Expands to the pid of the MPI starter process. (This process is the one that hit the MemoryScape MPIR_Breakpoint .) It can also be the process to which you manually attach. If no pid is available, %I expands to 0.
%J	Expands to the job ID. For MPICH or poe jobs, is the contents of the totalview_jobid variable contained either in the starter or first process. If that variable does not exist, it is set to zero (“0”). If it is not appropriate for the kind of job being launched, its value is -1.
%L	If MemoryScape is launching one process, this is replaced by the host name and TCP/IP port number (<i>hostname:port</i>) on which MemoryScape is listening for connections from tvdsvr .
%N	Is replaced by the number of servers that MemoryScape will launch. This is only used in a bulk server launch command.
%P	If MemoryScape is launching one process, this is replaced by the password that it automatically generated.
%R	Is replaced by the host name of the remote machine.
%S	If MemoryScape is launching one process, it replaces this symbol with the port number on the machine upon which MemoryScape is running.

%V	<p>If a bulk server launch is being performed, MemoryScape replaces this with a comma-separated list of port numbers. U</p> <p>(Sun) Expands to the local socket ID.</p> <p>Is replaced by the current MemoryScape verbosity setting.</p>	%Z	<p>Expands to the job ID. For MPICH or poe jobs, is the contents of the totalview_jobid variable contained either in the starter or first process. If that variable does not exist, it is set to zero ("0"). If it is not appropriate for the kind of job being launched, its value is -1.</p>
----	---	----	---

Index

Symbols

%B server launch replacement character 9
%C server launch replacement character 9
%D path name replacement character 9
%L host and port replacement character 9
%N line number replacement character 9
%P password replacement character 9
%S source file replacement character 9
%V verbosity setting replacement character 10

A

arguments
 for tvdsvr command 7

B

backtrace_depth keyword 6
backtrace_trim keyword 6
backtraces
 setting depth 6
block painting keywords 1

C

-callback option 7
-callback_host 7

-callback_ports 8
connection directory 9
console output for tvdsvr 8

D

-debug_file option 8
depth, backtraces 6
display_allocations_on_exit keyword 6

E

enable_event_filtering keyword 2
enable_guard_blocks keyword 4
enable_hoarding keyword 5
event reporting keywords 2

F

fifo hoard queue 5

G

guard_post_pattern keyword 5
guard_post_size keyword 4
guard_pre_pattern keyword 4
guard_pre_size keyword 4

H

hoard capacity 5
-hoard option 5
hoard_maximum_kb keyword 5
hoard_maximum_num_blocks keyword 5
hoarding 5
host ports 7
hostname
 for tvdsvr 7
 replacement 9

K

keywords
 block painting 1
 event reporting 2
 guard blocks 4
 hoarding 5
 logging 5

M

memalign_strict_alignment_even_multiple keyword 6
memory, reusing 5

W

N

- naming the host 7
- notify_addr_not_at_start keyword 3
- notify_alloc_not_in_heap keyword 3
- notify_alloc_null keyword 3
- notify_alloc_returned_bad_alignment keyword 3
- notify_bad_alignment_argument keyword 3
- notify_dealloc keyword 4
- notify_double_alloc keyword 3
- notify_double_dealloc keyword 3
- notify_free_not_allocated keyword 3
- notify_guard_corruption keyword 3
- notify_realloc keyword 4
- notify_realloc_not_allocated keyword 3
- notify_termination keyword 3

O

- option file 1
- options
 - tvdsvr
 - callback 7
 - serial 7
 - server 7
 - set_pw 7
- output_fd keyword 5
- output_file keyword 5

P

- paint option 1
- paint_alloc_pattern keyword 2
- paint_dealloc_pattern keyword 2
- paint_on_alloc keyword 2
- paint_on_dealloc keyword 2
- paint_on_zalloc keyword 2
- painting 1
- password checking 8
- passwords 8
 - generated by tvdsvr 7
- PATH environment variable
 - for tvdsvr 7
- port 4142 8
- port number 8

- for tvdsvr 7
- replacement 9
- searching 8
- port option 8
- ports on host 7

R

- remsh command 9
- reusing memory 5

S

- search_port option 8
- serial option 7
- server launch command 9
- server option 7, 8
- servers, number of 9
- set_pw option 7, 8

T

- tv_stop_on_event keyword 2
- tvdsvr command 7, 9
 - description 7
 - options 7
 - password 7
 - PATH environment variable 7
 - synopsis 7
- tvdsvr.conf 8
- TVDSVRLAUNCHCMD environment variable 9
- TVHEAP_ARGS file 1
- TVHEAP_ARGS variable 1
- TVHEAP_CONFIG_FILE variable 1

V

- verbosity keyword 5
- verbosity option 8
- verbosity setting replacement character 10

W

- working_directory option 8