

MEMORYSCAPE

**CREATING
PROGRAMS**



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 Creating Programs for Memory Debugging

Compiling Programs	1	Performing a Remote Login	9
Linking with the dbfork Library	2	Starting MemoryScape on a PE Program	9
dbfork on IBM AIX on RS/6000 Systems	2	Attaching to a PE Job	9
Linking C++ Programs with dbfork	3	Debugging LAM/MPI Applications	10
dbfork and Linux or Mac OS X	3	Debugging QSW RMS Applications	10
dbfork and SunOS 5 SPARC	3	Starting TotalView Debugger on an RMS Job	10
Starting MemoryScape	4	Attaching to an RMS Job	10
Attaching to Programs	5	Debugging Sun MPI Applications	11
Setting Up MPI Debugging Sessions	5	Attaching to a Sun MPI Job	11
Debugging MPI Programs	5	Linking Your Application with the Agent	11
Debugging MPICH Applications	6	Using env to Insert the Agent	13
Starting MemoryScape on an MPICH Job	7	Installing tvheap_mr.a on AIX	13
Attaching to an MPICH Job	7	LIBPATH and Linking	14
Using MPICH P4 procgroup Files	8	Using MemoryScape in Selected Environments	15
Starting MPI Issues	8	MPICH	15
Debugging IBM MPI Parallel Environment (PE) Applications	8	IBM PE	15
Using Switch-Based Communications	9	RMS MPI	15

Creating Programs for Memory Debugging

MemoryScape tries hard to cover up the details involved in starting your program. The way in which development and production environments are set up means that our defaults won't always do.

This chapter contains information you'll need if our defaults do not meet your need. Topics in this chapter are:

- "*Compiling Programs*" on page 1
- "*Linking with the dbfork Library*" on page 2
- "*Starting MemoryScape*" on page 4
- "*Attaching to Programs*" on page 5
- "*Setting Up MPI Debugging Sessions*" on page 5
- "*Linking Your Application with the Agent*" on page 11
- "*Using env to Insert the Agent*" on page 13
- "*Installing tvheap_mr.a on AIX*" on page 13
- "*Using MemoryScape in Selected Environments*" on page 15

Compiling Programs

The first step in getting a program ready for MemoryScape is to add your compiler's `-g` debugging command-line option. This option tells your compiler to generate symbol table debugging information; for example:

```
cc -g -o executable source_program
```

You can also use MemoryScape on programs that you did not compile using the `-g` option, or programs for which you do not have source code. However, MemoryScape may not be able to provide source code information.

The following table presents some general considerations

Compiler Option or Library	What It Does	When to Use It
Debugging symbols option (usually <code>-g</code>)	Generates debugging information in the symbol table.	Before debugging <i>any</i> program with MemoryScape.
Optimization option (usually <code>-O</code>)	Rearranges code to optimize your program's execution. Some compilers won't let you use the <code>-O</code> option and the <code>-g</code> option at the same time. Even if your compiler lets you use the <code>-O</code> option, don't use it when debugging your program, since strange results often occur.	After you finish debugging your program.
Multiprocess programming library (usually <code>dbfork</code>)	Uses special versions of the <code>fork()</code> and <code>execve()</code> system calls. In some cases, you need to use the <code>-lpthread</code> option. For more information about <code>dbfork</code> , see " Linking with the dbfork Library " in this chapter.	Before debugging a multiprocess program that explicitly calls <code>fork()</code> or <code>execve()</code> .

Linking with the dbfork Library

If your program uses the `fork()` and `execve()` system calls, and you want to debug the child processes, you need to link programs with the `dbfork` library.



While you must link programs that use `fork()` and `execve()` with the `dbfork` library so that MemoryScape can automatically attach to them when your program creates them, programs that you attach to need not be linked with this library.

dbfork on IBM AIX on RS/6000 Systems

Add either the `-dbfork` or `-ldbfork_64` argument to the command that you use to link your programs. If you are compiling 32-bit code, use the following arguments:

- `/memscape_install_dir/lib/libdbfork.a \`
`-bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a`
- `-L/memscape_install_dir/lib \`
`-ldbfork -bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a`

For example:

```
cc -o program program.c \
-L/usr/totalview/rs6000/lib/ -ldbfork \
-bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

If you are compiling 64-bit code, use the following arguments:

- `/memscape_install_dir/lib/libdbfork_64.a \`
`-bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a`
- `-L/memscape_install_dir/lib -ldbfork_64 \`
`-bkeepfile:/usr/totalviewrs6000/lib/libdbfork.a`

For example:

```
cc -o program program.c \
-L/usr/totalview/rs6000/lib -ldbfork \
-bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

When you use `gcc` or `g++`, use the `-Wl,-bkeepfile` option instead of using the `-bkeepfile` option, which will pass the same option to the binder. For example:

```
gcc -o program program.c \
-L/usr/totalview/rs6000/lib -ldbfork -Wl, \
-bkeepfile:/usr/totalview/rs6000/lib/libdbfork.a
```

Linking C++ Programs with dbfork. You cannot use the `-bkeepfile` binder option with the IBM xLC C++ compiler. The compiler passes all binder options to an additional pass called **munch**, which will not handle the `-bkeepfile` option.

To work around this problem, we have provided the C++ header file `libdbfork.h`. You must include this file somewhere in your C++ program. This forces the components of the **dbfork** library to be kept in your executable. The file `libdbfork.h` is included only with the RS/6000 version of MemoryScape. This means that if you are creating a program that will run on more than one platform, you should place the `include` within an `#ifdef` statement's range. For example:

```
#ifdef _AIX
#include "/usr/totalview/include/libdbfork.h"
#endif
int main (int argc, char *argv[])
{
}
```

In this case, you would not use the `-bkeepfile` option and would instead link your program using one of the following options:

- `/usr/totalview/include/libdbfork.a`
- `-L/usr/totalview/include -ldbfork`

dbfork and Linux or Mac OS X

Add one of the following arguments or command-line options to the command that you use to link your programs:

- `/usr/totalview/platform/lib/libdbfork.a`
- `-L/usr/totalview/platform/lib -ldbfork`
or
`-L/usr/totalview/platform/lib -ldbfork_64`

where *platform* is one of the following: `darwin-power`, `linux-x86`, `linux-x86-64`, `linux-ia64`, or `linux-power`.

In general, 32-bit programs use `libdbfork.a` and 64-bit programs use `libdbfork_64.a`. Of course, if your architecture doesn't support 32-bit programs, the option won't work.

For example:

```
cc -o program program.c \
-L/usr/totalview/linux-x86/lib -ldbfork
```

However, `linux-ia64` uses `libdbfork` for 64-bit programs.

dbfork and SunOS 5 SPARC

Add one of the following command line arguments or options to the command that you use to link your programs:

- `/opt/totalview/sun5/lib/libdbfork.a`
- `-L/opt/totalview/sun5/lib -ldbfork`

For example:

```
cc -o program program.c \
-L/opt/totalview/sun5/lib -ldbfork
```

As an alternative, you can set the `LD_LIBRARY_PATH` environment variable and omit the `-L` option on the command line:

```
setenv LD_LIBRARY_PATH /opt/totalview/sun5/lib
```

Starting MemoryScape

MemoryScape can debug programs that run in many different computing environments and which use many different parallel processing modes and systems. This section looks at few of the ways you can start MemoryScape.

In most cases, the command for starting MemoryScape looks like the following:

```
memscape [ executable [ corefile ] ] [ options ]
```

where *executable* is the name of the executable file and *corefile* is the name of the core file that you want to examine.

Your environment may require you to start MemoryScape in another way. For example, if you are debugging an MPI program, you may need to invoke MemoryScape on `mpirun`. Details are presented later in this chapter.

The following examples show different ways of starting MemoryScape:

Starting MemoryScape

```
memscape
```

Starts MemoryScape without loading a program or core file. You now select links such as **Add new program** or **Add parallel program** links to tell MemoryScape to load a program.

Starting MemoryScape and Naming a program

```
memscape executable
```

Starts MemoryScape and loads the *executable* program.

Using core files

```
memscape executable corefile
```

Starts MemoryScape and loads the *executable* program and the *corefile* core file.

Passing arguments to the program

```
memscape executable -a args
```

Starts MemoryScape and passes all the arguments following the `-a` option to the *executable* program.

When you use the `-a` option, you must enter it as the last MemoryScape option on the command line.

If you don't use the `-a` option and you want to add arguments after MemoryScape loads your program, right click on the executable and select Properties.

Debugging a program that runs on another computer

```
memscape executable -remote hostname_or_address[:port]
```

Starts MemoryScape on your local host and the TotalView Debugger Server (**tvdsvr**) on a remote host. After MemoryScape begins executing, it loads the program specified by *executable* for remote debugging. You can specify a host name or a TCP/IP address. If you need to, you can also enter the TCP/IP port number.

If MemoryScape fails to automatically load a remote executable, you may need to disable *autolaunching* for this connection and manually start the Debugger Server (**tvdsvr**). (*Autolaunching* is the process of automatically launching **tvdsvr** processes.) You can disable autolaunching by adding the *hostname:portnumber* suffix to the name you type in the **Host** field of the **Add new program** or **Add parallel program** screens. As always, the *portnumber* is the TCP/IP port number on which our server is communicating with MemoryScape.

Debugging an MPI Program

```
memscape executable
```

(*method 1*) In many cases, you will start an MPI program in much the same way as you would start any other program. However, you will need to set its properties. One way of doing this is by selecting the executable's name from within MemoryScape, right-clicking, and selecting **Properties**. In the displayed dialog box, select the MPI version in addition to other options.

`mpirun -np count -tv executable`

(method 2) The MPI `mpirun` command starts the MemoryScape pointed to by the **TOTALVIEW** environment variable. MemoryScape then starts your program. This program will run using **count** processes.

Attaching to Programs

If a program you're testing is using too much memory, you can attach to it while it is running. MemoryScape lets you attach to single and multi-process programs, and these programs can be running remotely.

To attach to a process, select the **Attach to running program** item within the **Home | Add Program** page.



When you exit from MemoryScape, it kills all programs and processes that it started. However, programs and processes that were executing before you brought them under MemoryScape's control continue to execute.

If you want MemoryScape to automatically attach to programs that use `fork()` and `execve()`, you must use the MemoryScape `dbfork` library. However, programs that you manually attach to need not be linked with this library.

Setting Up MPI Debugging Sessions

This section explains how to set up MemoryScape MPI debugging sessions. The contents of this section is as follows:

- “*Debugging MPI Programs*” on page 5
- “*Debugging MPICH Applications*” on page 6
- “*Debugging IBM MPI Parallel Environment (PE) Applications*” on page 8
- “*Debugging IBM MPI Parallel Environment (PE) Applications*” on page 8
- “*Debugging LAM/MPI Applications*” on page 10
- “*Debugging QSW RMS Applications*” on page 10
- “*Debugging Sun MPI Applications*” on page 11

Debugging MPI Programs

In many cases, the way in which you invoke an MPI program within MemoryScape control differs little from discipline to discipline. If you invoke MemoryScape from the command line without an argument, MemoryScape displays its **Add Programs to Your MemoryScape Session** screen. Select **Add parallel program** and then fill in the information presented in the various screens.

For example, you should select the **Parallel system**, the number of **Tasks**, and possibly **Nodes**. If there are additional arguments that need to be sent to the starter process, type them within the **MPI launcher arguments** area. These arguments are ones that are sent to a starter process such as `mpirun` or `poe`. They are not arguments sent to your program, which are instead entered within the **Command line** arguments area. If you need to add environment variables, enter them within the **Environment variables** tab.

In most cases, MemoryScape will remember what you type between invocations of MemoryScape. For example, suppose you were debugging a program called `my_foo` and set it up using these controls. The next time you start MemoryScape upon this program, this page will show the last entered entries after you select the program's name from **Name of the program** pulldown.

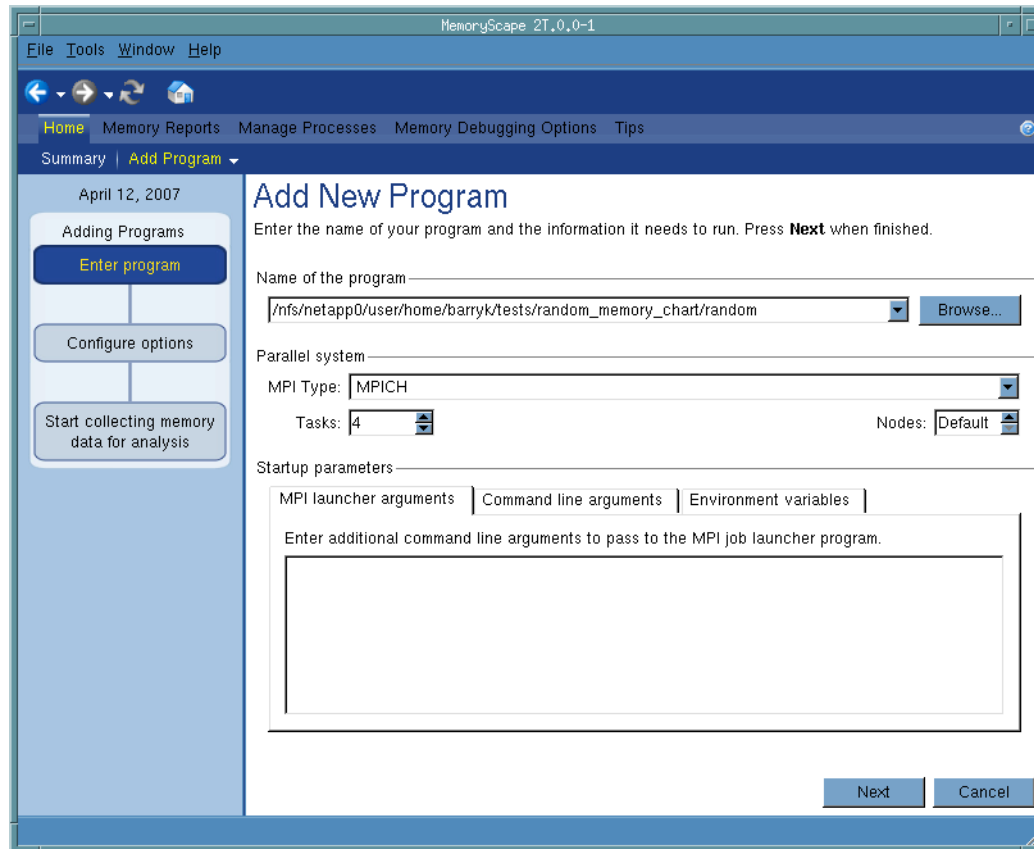


Figure 1: Adding a Parallel program

Debugging MPICH Applications



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

To examine Message Passing Interface/Chameleon Standard (MPICH) applications, you must use MPICH version 1.2.5 or later on a homogeneous collection of computers. If you need a copy of MPICH, you can

obtain it at no cost from Argonne National Laboratory at www.mcs.anl.gov/mpi. (We strongly urge that you use a later version of MPICH.)

The MPICH library should use the `ch_p4`, `ch_p4mpd`, `ch_shmem`, `ch_lfshmem`, or `ch_mpl` devices.

- For networks of workstations, the default MPICH library is `ch_p4`.
- For shared-memory SMP computers, use `ch_shmem`.
- On an IBM SP computer, use the `ch_mpl` device.

The MPICH source distribution includes all of these devices. Choose the device that best fits your environment when you configure and build MPICH.

For more information, see:

- Starting MemoryScape on an MPICH Job
- “Attaching to an MPICH Job” on page 7
- “Using MPICH P4 *procgroup* Files” on page 8

Starting MemoryScape on an MPICH Job. Before you can bring an MPICH job under MemoryScape’s control, both MemoryScape and the Debugger Server must be in your path. You can do this in a login or shell startup script.

The *official* syntax for starting Memoryscape is as follows:

```
mpirun -tv [ other_mpich-args ] program [ program-args ]
```

For example:

```
mpirun -tv -np 4 sendrecv
```

The `-tv` option tells `mpirun` that it should obtain information from the `TOTALVIEW` environment variable.

For example, the following is the C shell command that sets the `TOTALVIEW` environment variable so that `mpirun` passes the `-verbosity` option to MemoryScape:

```
setenv TOTALVIEW "memscape -verbosity 1"
```

In this example, the `memscape` command must be in your path. If it isn’t, you’ll need to specify either an absolute or relative path to the `memscape` command. MemoryScape begins by starting the first process of your job, the master process, under its control.

On the IBM SP computer with the `ch_mpl` device, the `mpirun` command uses the `poe` command to start an MPI job. While you still must use the MPICH `mpirun` (and its `-tv` option) command to start an MPICH job, the way you start MPICH differs. For details on using MemoryScape with `poe`, see “Starting MemoryScape on a PE Program” on page 9.

Starting MemoryScape using the `ch_p4mpd` device is similar to starting MemoryScape using `poe` on an IBM computer or other methods you might use on Sun and HP platforms. In general, you start MemoryScape using the `memscape` command, with the following syntax;

```
memscape mpirun [ memscape_args ] -a [ mpich-args ] \  
program [ program-args ]
```

As your program executes, MemoryScape automatically acquires the processes that are part of your parallel job as your program creates them. MemoryScape automatically copies memory configuration information to the slave processes.

Attaching to an MPICH Job. MemoryScape lets you to attach to an MPICH application even if it was not started under MemoryScape’s control. These processes, however, much have previously been linked with the MemoryScape agent. See for “Linking Your Application with the Agent” on page 11 for details.

To attach to an MPICH application:

1 Start MemoryScape.

Select **Attach to running program** from the **Add Programs to Your MemoryScape Session** screen. You are now shown processes that are not yet owned.

2 Attach to the first MPICH process in your workstation cluster by selecting it then clicking Next.

3 On an IBM SP with the `ch_mpi` device, attach to the `poe` process that started your job. For details, see “Starting MemoryScape on a PE Program” on page 9. The following figure shows this information.

- Normally, the first MPICH process is the highest process with the correct program name in the process list. Other instances of the same executable can be:
- The `p4` listener processes if MPICH was configured with `ch_p4`.
- Additional slave processes if MPICH was configured with `ch_shmem` or `ch_lfshmem`.

Setting Up MPI Debugging Sessions

- Additional slave processes if MPICH was configured with `ch_p4` and has a file that places multiple processes on the same computer.

4 After you attach to your program's processes, MemoryScape asks if you also want to attach to slave MPICH processes. If you do, press **Return** or choose **Yes**. If you do not, choose **No**.

If you choose **Yes**, MemoryScape starts the server processes and acquires all MPICH processes.

In some situations, the processes you expect to see might not exist (for example, they may crash or exit). MemoryScape acquires all the processes it can and then warns you if it can not attach to some of them. If you attempt to dive into a process that no longer exists (for example, using a message queue display), MemoryScape tells you that the process no longer exists.

Using MPICH P4 procgroup Files. If you're using MPICH with a P4 `procgroup` file (by using the `-p4pg` option), you must use the *same* absolute path name in your `procgroup` file and on the `mpirun` command line. For example, if your `procgroup` file contains a different path name than that used in the `mpirun` command, even though this name resolves to the same executable, MemoryScape assumes that it is a different executable, which causes debugging problems.

The following example uses the same absolute path name on the MemoryScape command line and in the `procgroup` file:

```
% cat p4group
local 1 /users/smith/mympichexe
bigiron 2 /users/smith/mympichexe
% mpirun -p4pg p4group -tv /users/smith/mympichexe
```

In this example, MemoryScape does the following:

- 1 Reads the symbols from `mympichexe` only once.
- 2 Places MPICH processes in the same MemoryScape share group.
- 3 Names the processes `mympichexe.0`, `mympichexe.1`, `mympichexe.2`, and `mympichexe.3`.

If MemoryScape assigns names such as `mympichexe<mympichexe>.0`, a problem occurred and you need to compare the contents of your `procgroup` file and `mpirun` command line.

Starting MPI Issues



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

If you can't successfully start MemoryScape on MPI programs, check the following:

- Can you successfully start MPICH programs without MemoryScape? The MPICH code contains some useful scripts that let you verify that you can start remote processes on all of the computers in your `computers` file. (See `tstmachines` in `mpich/util`.)
- Does the TotalView Server (`tvdsvr`) fail to start? Remember that MemoryScape uses `rsh` to start the server, and that this command doesn't pass your current environment to remotely started processes.
- Under some circumstances, MPICH kills MemoryScape with the `SIGINT` signal. You can see this behavior when you use the `Kill` command as the first step in restarting an MPICH job.

Debugging IBM MPI Parallel Environment (PE) Applications



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

You can debug IBM MPI Parallel Environment (PE) applications on the IBM RS/6000 and SP platforms.

To take advantage of MemoryScape's ability to automatically acquire processes, you must be using release 3,1 or later of the Parallel Environment for AIX.

Topics in this section are:

- "Starting MemoryScape on a PE Program" on page 9
- "You should start all of your parallel tasks using the **Run** command." on page 9
- "Attaching to a PE Job" on page 9

The following sections describe what you must do before MemoryScape can debug a PE application.



You must link the MemoryScape agent into your IBM MPI Parallel Environment program before running it. For details, see "Installing tvheap_mr.a on AIX" on page 13.

Using Switch-Based Communications. If you're using switch-based communications (either *IP over the switch* or *user space*) on an SP computer, you must configure your PE debugging session so that MemoryScape can use *IP over the switch* for communicating with the TotalView Server (tvdsvr). Do this by setting the `-adapter_use` option to **shared** and the `-cpu_use` option to **multiple**, as follows:

- If you're using a PE host file, add **shared multiple** after all host names or pool IDs in the host file.
- Always use the following arguments on the **poe** command line:
`-adapter_use shared -cpu_use multiple`

If you don't want to set these arguments on the **poe** command line, set the following environment variables before starting **poe**:

```
setenv MP_ADAPTER_USE shared
setenv MP_CPU_USE multiple
```

When using *IP over the switch*, the default is usually **shared adapter use** and **multiple cpu use**; we recommend that you set them explicitly using one of these techniques. You must run MemoryScape on an SP or SP2 node. Since MemoryScape will be using *IP over the switch* in this case, you cannot run MemoryScape on an RS/6000 workstation.

Performing a Remote Login. You must be able to perform a remote login using the **rsh** command. You also need to enable remote logins by

adding the host name of the remote node to the `/etc/hosts.equiv` file or to your `.rhosts` file.

When the program is using switch-based communications, MemoryScape tries to start the TotalView Server by using the **rsh** command with the switch host name of the node.

Setting Timeouts

If you receive communications timeouts, you can set the value of the `MP_TIMEOUT` environment variable; for example:

```
setenv MP_TIMEOUT 1200
```

If this variable isn't set, the default **timeout** value is 600 seconds.

Starting MemoryScape on a PE Program. The following is the syntax for running Parallel Environment (PE) programs from the command line:

```
program [ arguments ] [ pe_arguments ]
```

You can also use the **poe** command to run programs as follows:

```
poe program [ arguments ] [ pe_arguments ]
```

If, however, you start MemoryScape on a PE application, you must start **poe** as MemoryScape's target using the following syntax:

```
memscape poe -a program [ arguments ] [ PE_arguments ]
```

For example:

```
memscape poe -a sendrecv 500 -rmpool 1
```

You should start all of your parallel tasks using the **Run** command.



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

Attaching to a PE Job. To take full advantage of MemoryScape's **poe**-specific automation, you need to attach to **poe** itself, and let MemoryScape automatically acquire the **poe** processes on all of its

nodes. In this way, MemoryScape acquires the processes you want to debug.

Attaching from a Node Running poe

To attach MemoryScape to **poe** from the node running **poe**:

- 1 Start MemoryScape in the directory of the debug target.

If you can't start MemoryScape in the debug target directory, you can start MemoryScape by editing the Debugger Server (**tvdsvr**) command line before attaching to **poe**.

- 2 In the **Home | Add Program** screen, select **Attach to running program**, then find the **poe** process list, select it, then hit the **Next** button. When necessary, MemoryScape launches **tvdsvr** processes.
- 3 Locate the process you want to debug and dive on it.

If your source code files are not displayed in the Source Pane, you might not have told MemoryScape where these files reside. You can fix this by invoking the **File > Search Path** command to add directories to your search path.

Attaching from a Node Not Running poe

The procedure for attaching MemoryScape to **poe** from a node that is not running **poe** is essentially the same as the procedure for attaching from a node that is running **poe**.

To place **poe** in this list:

- 1 Connect MemoryScape to the startup node.
- 2 From the **Home | Add Programs to Your MemoryScape Session** page, select **Attach to running program**.
- 3 Look for the process named **poe** and continue as if attaching from a node that is running **poe**.

Debugging LAM/MPI Applications



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

You debug a LAM/MPI program in a similar way to how you debug most MPI programs. Use the following syntax if MemoryScape is in your path:

```
mpirun -tv mpirun args prog prog_args
```

As an alternative, you can invoke MemoryScape on **mpirun**:

```
memscape mpirun -a prog prog_args
```

Debugging QSW RMS Applications



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

MemoryScape supports automatic process acquisition on AlphaServer SC systems and 32-bit Red Hat Linux systems that use Quadrics RMS resource management system with the QSW switch technology.

Starting TotalView Debugger on an RMS Job. To start a parallel job under MemoryScape's control, use MemoryScape as if you were debugging **prun**:

```
memscape prun -a prun-command-line
```

MemoryScape starts and shows you the machine code for RMS **prun**. Since you're not usually interested in debugging this code, use the **Run** command to let the program run.

The RMS **prun** command executes and starts all MPI processes.

Attaching to an RMS Job. To attach to a running RMS job, attach to the RMS **prun** process that started the job.

You attach to the **prun** process the same way you attach to other processes.

Debugging Sun MPI Applications



In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 5.

MemoryScope can debug a Sun MPI program and can display Sun MPI message queues. This section describes how to perform *job startup* and *job attach* operations.

To start a Sun MPI application, use the following procedure, type the following command:

```
memscape mprun [ totalview_args ] -a [ mpi_args ]
```

For example:

```
memscape mprun -g blue -a -np 4 /usr/bin/mpi/conn.x
```

When the MemoryScope Window appears, select the **Go** button.

Attaching to a Sun MPI Job. To attach to an already running **mprun** job:

- 1 Find the host name and process identifier (PID) of the **mprun** job by typing **mpps -b**. For more information, see the **mpps(1M)** manual page.

The following is sample output from this command:

JOBNAME	MPRUN_PID	MPRUN_HOST
cre.99	12345	hpc-u2-9
cre.100	12601	hpc-u2-8

- 2 Go to **Attach to Running Program** and look for the process.

- 3 If MemoryScope is running on a different node than the **mprun** job, enter the host name in the **Remote Host** field.



You must link the MemoryScope agent into your Sun MPI Parallel Environment program before running it. For details, see "Linking Your Application with the Agent" on page 11.

Linking Your Application with the Agent

MemoryScope puts its heap agent between your program and its heap library. This allows the agent to intercept the calls that your program makes to this library. After it intercepts the call, it checks the call for errors and then sends it on to the library so that it can be processed. The MemoryScope agent does not replace standard memory functions; it just monitors what they do.

In most cases, you do not have to link your application with the heap agent's library. If this is the case, you can ignore the rest of the information in this chapter.

In some situations, you need to explicitly link the MemoryScope agent directly to your program such as when attaching to a file or debugging core file. For example, if you are debugging an MPI program, your starter program might not propagate environment variables if you name a starter process such as **mpirun** on the command line.



On Apple Mac OS X, you cannot link the agent into your program.

You need to incorporate the agent into your environment in either of the following ways:

- Linking your application with the agent.
- Requesting that the agent's library be preloaded by setting a runtime loader environment variable. This is only done when MemoryScope or your program will attach to another program that it did not start

and you want MemoryScape to locate problems in this second application.

AIX applications differ from applications running on other platforms as AIX does not support interposition. However,

MemoryScape can replace the AIX heap library. See "Installing tvheap_mr.a on AIX" on page 13.

The following table lists additional linker options that you must use when you link your program:

Platform	Compiler	Binary Interface	Additional linker options
HP Tru64 Alpha (version 5)	Compaq/KCC	64	-Lpath -ltvheap -rpath path
	GCC	64	-Lpath -ltvheap -Wl,-rpath,path
IBM RS/6000 (all)	IBM/GCC	32/64	-Lpath_mr -Lpath
	KCC	32	-Lpath_mr -Lpath --static_libKCC
		64	-Lpath_mr -Lpath
AIX 5	IBM/GCC/KCC	32	-Lpath_mr -Lpath path/aix_malloctype.o
		64	-Lpath_mr -Lpath path/aix_malloctype64_5.o
Linux x86	GCC/Intel/PGI	32	-Lpath -ltvheap -Wl,-rpath,path
	KCC	32	-Lpath -ltvheap -rpath path
Linux x86-64	GCC/PGI	32	-Lpath -ltvheap -Wl,-rpath,path
		64	-Lpath -ltvheap_64 -Wl,-rpath,path
Linux IA64	GCC/Intel	64	-Lpath -ltvheap -Wl,-rpath,path
Sun	Sun/KCC/Apogee	32	-Lpath -ltvheap -R path
	Sun/KCC	64	-Lpath -ltvheap_64 -R path
	GCC	32	-Lpath -ltvheap -Wl,-R,path
		64	-Lpath -ltvheap_64 -Wl,-R,path

The following list describes the options in this table:

- path* The absolute path to the agent in the MemoryScape installation hierarchy. More precisely, this directory is:
installdir/toolworks/totalview.version/platform/lib
- installdir* The installation base directory name.
- version* The MemoryScape version number.
- platform* The platform tag.
- path_mr* The absolute path of the heap replacement library. This value is determined by the person who installs the MemoryScape malloc replacement library.

Since it is easy to misinterpret the path specifications, you may want to see what value MemoryScape uses when it sets a path. Here's the procedure:

- 1 Start MemoryScape. You'll need to start MemoryScape on a program, but it need not be the program you are debugging.
- 2 Right-click on the filename and select **Properties**. In the displayed dialog box, select the **Environment** Page. Type a value that is the same as or similar to the one in Figure 2.



Installing tvheap_mr.a on AIX requires that the system have the bos.adt.syscalls System Calls Application Toolkit page installed.

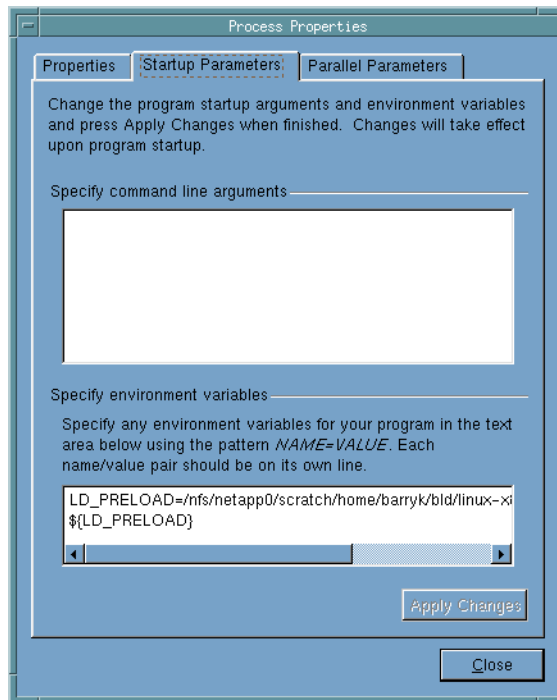


Figure 2: Interposition Path

Using env to Insert the Agent

When MemoryScape attaches to a process that is already running, the agent must already be associated with it. You can do this in two ways:

- Manually link the agent as is described in previous sections.
- Start the program using `env`. This pushes the agent into your program.

The variable that you use must on each platform are shown in the following table:

Platform	Variable
Apple Mac OS X	DYLD_INSERT_LIBRARIES
HP Tru64 Alpha	_RLD_LIST
IBM AIX	MALLOCTYPE
Linux IA64 and x86	LD_PRELOAD
SGI Irix	__RLDN32_LIST _RLD64_LIST
Sun	LD_PRELOAD

You can display the value that MemoryScape uses by displaying the **Environment** Page within the **Properties** dialog box. To set this variable:

- 1 Start MemoryScape. You'll need to start MemoryScape on a program, but it need not be the programming you are debugging.
- 2 Right-click on the filename and select **Properties**. If you now select the **Environment** tab, you'll see what the value for your environment is.
- 3 Close MemoryScape.
- 4 Start the program to which you will be attaching as an argument to the `env` command. For example, here's how to set this variable on AIX:


```
env MALLOCTYPE=user:tvheap_mr.a memscape my_prog
```

Installing tvheap_mr.a on AIX

You must install the `tvheap_mr.a` library on each node upon which you will be running the MemoryScape agent. The `aix_install_tvheap_mr.sh` script contains most of what you need to do. This script is in the following directory:

```
toolworks/totalview.version/rs6000/lib/
```

Installing tvheap_mr.a on AIX

For example, after you become **root**, enter the following commands:

```
cd toolworks/memscape.1.0.0-0/rs6000/lib
mkdir /usr/local/tvheap_mr \
./aix_install_tvheap_mr.sh ./tvheap_mr.tar \
/usr/local/tvheap_mr
```

Use **poe** to create **tvheap_mr.a** on multiple nodes.

The pathname for the **tvheap_mr.a** library must be the same on each node. This means that you cannot install this library on a shared file system. Instead, you must install it on a file system that is private to the node. For example, because **/usr/local** is usually only accessible from the node upon which it is installed, you might want to install it there.



*The **tvheap_mr.a** library depends heavily on the exact version of **libc.a** that is installed on a node. If **libc.a** changes, you must recreate **tvheap_mr.a** by re-executing the **aix_install_tvheap_mr.sh** script.*

If this malloc replacement library changes—it doesn't change very often—you'll need to rerun this procedure. Our new features will tell you about this change.

LIBPATH and Linking

This section discusses compiling and linking your AIX programs. The following command adds **path_mr** and **path** to your program's libpath:

```
xlc -Lpath_mr -Lpath -o a.out foo.o
```

When **malloc()** dynamically loads **tvheap_mr.a**, it should find the library in **path_mr**. When **tvheap_mr.a** dynamically loads **tvheap.a**, it should find it in **path**.

The AIX linker allows you to relink executables. This means that you can make an already complete application ready for the MemoryScape agent; for example:

```
cc a.out -Lpath_mr -Lpath -o a.out.new
```

Here's an example that does not link in the heap replacement library. Instead, it allows you to dynamically set **MALLOCTYPE**:

```
xlc -q32 -g \
-L/usr/local/tvheap_mr \
-L/home/memscape/interposition/lib prog.o -o prog
```

The next example shows you how to allow your program to access the MemoryScape agent by linking in the **aix_malloctype.o** module:

```
xlc -q32 -g \
-L/usr/local/tvheap_mr \
-L/home/memscape/interposition/lib prog.o \
/home/memscape/interposition/lib/aix_malloctype.o \
-o prog
```

You can check that the paths made it into the executable by running the **dump** command; for example:

```
% dump -Xany -Hv tx_memdebug_hello
```

```
tx_memdebug_hello:
```

```
***Loader Section***
Loader Header Information
VERSION#  #SYMtableENT  #RELOCent  LENidSTR
0x00000001 0x0000001f  0x00000040 0x000000d3

#IMPfilID  OFFidSTR  LENstrTBL  OFFstrTBL
0x00000005 0x00000608 0x00000080 0x000006db

***Import File Strings***
INDEX  PATH  BASE  MEMBER
0  /.../lib:/usr/.../lib:/usr/lib:/lib
1  libc.a  shr.o
2  libC.a  shr.o
3  libpthreads.a  shr_comm.o
4  libpthreads.a  shr_xpg5.o
```

Index 0 in the **Import File Strings** section shows the search path the runtime loader uses when it dynamically loads a library. Some systems propagate the preload library environment to the processes they will run; others, do not. If they do not, you need to manually link them with the **tvheap** library.

In some circumstances, you might want to link your program instead of setting the **MALLOCTYPE** environment variable. If you set the

`MALLOCTYPE` environment variable for your program and it uses `fork()/exec()` a program that is not linked with the agent, your program will terminate because it fails to find `malloc()`.

Using MemoryScape in Selected Environments

This topic describes using the Memory Debugger within various environments. The sections within this topic are:

- MPICH
- IBM PE
- RMS MPI

MPICH

Here's how to use MemoryScape with MPICH MPI codes. TotalView Technologies has tested this only on Linux x86.

- 1 You must link your parallel application with the MemoryScape agent as described "LIBPATH and Linking" on page 14. On most Linux x86 systems, you'll type:

```
mpicc -g test.o -o test -Lpath \
    -ltvheap -Wl,-rpath,path
```

- 2 Start MemoryScape using the `-tv` command-line option to the `mpirun` script in the usual way. For example:

```
mpirun -tv mpirun-args test args
```

MemoryScape will start up on the rank 0 process.

- 3 If you need to configure MemoryScape, you should do it now.
- 4 Run the rank 0 process.

IBM PE

Here's how to use MemoryScape with IBM PE MPI codes:

- 1 You must prepare your parallel application to use the MemoryScape agent in "LIBPATH and Linking" on page 14 and in "Installing `tvheap_mr.a` on AIX" on page 13. Here is an example that usually works:

```
mpicc_r -g test.o -o test -Lpath_mr -Lpath \
    path/aix_malloctype.o
```

"Installing `tvheap_mr.a` on AIX" on page 13 contains additional information.

- 2 Start MemoryScape on `poe` as usual:

```
memscape poe -a test args
```

Because `tvheap_mr.a` is not in `poe`'s LIBPATH, enabling MemoryScape upon the `poe` process will cause problems because `poe` will not be able to locate the `tvheap_mr.a` malloc replacement library.

- 3 If you need to configure MemoryScape, you should do it now.
- 4 Run the `poe` process.

RMS MPI

Here's how to use MemoryScape with Quadrics RMS MPI codes. TotalView Technologies has tested this only on Linux x86.

- 1 There is no need to link the application with MemoryScape because `prun` propagates environment variables to the rank processes. However, if you'd like to link the application with the agent, you can.

- 2 Start MemoryScape on `prun`. For example:

```
memscape prun -a prun-args test args
```

- 3 If you need to configure MemoryScape, you should do it now.
- 4 Run the `prun` process.

Index

Symbols

__RLDN32_LIST heap debugging environment variable 13
_RLD_LIST heap debugging environment variable 13
_RLD64_LIST heap debugging environment variable 13

A

-a option 4
Add parallel program screen 5
Add Programs to Your MemoryScope Session screen 5, 7
agent and AIX 12
agent linking 11
agent, inseting with env 13
AIX
 compiling 64-bit code 2
 linking C++ to dbfork library 3
 linking to dbfork library 2
AIX issues 12
aix_install_tvheap_mr.sh script 13
arguments, remembering 5
Attach to running program screen 7
attaching to MPICH job 7
attaching to mprun job 11
attaching to PE jobs 9
attaching to programs 13

attaching to programss 5
attaching to Sun MPI job 11
autolaunching 4

B

-bkeepfile option 3

C

C++
 including libdbfork.h 3
Command line arguments area 5
commands, Process > Startup Parameters 13
compiling 64-bit code on AIX 2
compiling programs 1
core files, starting within MemoryScope 4

D

dbfork and Linux for Mac OS X 3
dbfork library
 linking with 2
debugging MPICH applicaitons 6

E

env, inserting agent 13
environment variables

LD_LIBRARY_PATH 3
Envrionment variables tab 5
execve() 2

F

files, libdbfork.h 3
fork() 2

H

heap debugging
 attaching to programs 13
 environment variable 13
 IBM PE 15
 incorporating agent 11
 linker command-line options 12
 MPICH 15
 RMS MPI 15
 setting environment variable 13
 tvheap_mr.a
 library 13
Host field 4
host names, specifying'specifiying host names 4

W

L

- LAM and MemoryScape 10
- LD_PRELOAD heap debugging environment variable 13
- libdbfork.a 2, 3
- libdbfork.h file 3
- libdbfork_64.a 3
- LIBPATH and linking 14
- linking agent 11
- linking and AIX 12
- linking to dbfork library 2
 - AIX 2
 - C++ and dbfork 3
 - SunOS 5 3

M

- Mac OS X, linking dbfork 3
- MALLOCTYPE heap debugging environment variable 13, 14
- MemoryScape and LAM 10
- MemoryScape and MPICH 7
- MemoryScape and PE 8
- MemoryScape and QSW RMS 10
- MemoryScape and Sun MPI 11
- memscape command 7
- MmemoryScape command line 4
- MPI
 - on Sun 11
- MPI issues 8
- MPI launcher arguments area 5
- MPI programs, starting 4
- MPICH
 - and heap debugging 15
- MPICH and MemoryScape 7
- MPICH and poe 7
- MPICH applications, debugging 6
- MPICH library, selecting 6
- MPICH P4 procgroup 8
- MPICH, attaching to job 7
- mpirun 4
- mprun 11
- mprun command 11
- mprun job, attaching to 11

N

- Nodes, selecting number of 5

P

- Parallel system, selecting 5
- passing arguments to programs 4
- PE and MemoryScape 8
- PE jobs, attaching 9
- poe and MPICH 7
- poe, starting using 9
- port number. 4
- preload variables, by platform 13
- Process > Startup Parameters command 13
- processes, attaching to 5
- programs
 - compiling 1
- programs, attaching to 5
- programs, passing arguments to 4
- prun, using 10

Q

- QSW RMS and MemoryScape 10

R

- remote 4
- remote computers, debugging on 4
- remote login 9
- remote programs, starting 4

S

- setting timeouts 9
- starting MemoryScape 4
- starting MPI programs 4
- starting remove programs 4
- starting with poe 9
- Sun MPI 11
- Sun MPI and MemoryScape 11
- SunOS 5
 - linking to dbfork library 3
- switch-based communications 9

T

- Tasks, selecting number of 5

- tasks, specifying 5
- timeouts, setting 9
- tv option 7
- tvdsrv 4
- tvheap_mr.a
 - aix_install_tvheap_mr.sh script 13
 - and aix_malloctype.o 14
 - creating using poe 14
 - dynamically loading 14
 - libc.a requirements 14
 - pathname requirements 14
 - relinking executables on AIX 14
- tvheap_mr.a library 13

U

- using env to insert agent 13
- using prun 10

W

- Wl,-bkeepfile option 3