

WIND RIVER

Wind River® Workbench Memory Analyzer

USER'S GUIDE

3.0

Copyright © 2007 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, the Wind River logo, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

<http://www.windriver.com/company/terms/trademark.html>

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at the following location:
installDir/product_name/3rd_party_licensor_notice.pdf.

Corporate Headquarters

Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

<http://www.windriver.com>

For information on how to contact Customer Support, please visit the following URL:

<http://www.windriver.com/support>

Contents

1	Introduction	1
1.1	Introduction	1
	Memory Allocation Problems	2
	Memory Analyzer Overview	2
1.2	Architectural Summary	3
	VxWorks Targets	4
	Linux Targets	4
	Wind River Linux 2.0 Targets	5
	Database Files	5
1.3	Features	6
2	Getting Started	9
2.1	Introduction	9
2.2	Requirements	10
	VxWorks	10
	Linux	12
2.3	Starting Memory Analyzer	13
	Initiating the Target Connection	14
	Opening the Memory Analyzer GUI	15

2.4	Testing Your Installation	16
	Viewing from the Shell	16
	Running the Demonstration Program	16
2.5	Usage Notes	19
	Symbol Resolution	19
	Patching	20
	Using memrun (Linux Only)	20
	Process Selection (Linux Only)	21
	Thread Analysis (Linux Only)	22
3	The Memory Analyzer GUI	23
3.1	Introduction	23
3.2	The Memory Analyzer GUI	24
3.2.1	Summary View	25
3.2.2	Aggregate View	26
	Aggregate Allocations Table	27
	Individual Allocations Table	30
	Aggregate View Pop-Up Menus	31
3.2.3	Tree View	31
	Call Stack Tree	32
	Individual Allocations Table	33
	Tree View Pop-Up Menus	33
3.2.4	Time View	34
	Graph Area	34
	Details Table	35
	Time View Pop-Up Menu	36
3.2.5	Fragmentation View (VxWorks Only)	36
	Fragmentation Map	37
	Individual Allocations Table	39
	Fragmentation View Pop-Up Menu	39
3.2.6	Details Viewport View	40
3.2.7	Source Code Viewer	41
	Call Stack Tree	41

	Details Viewport	42
3.2.8	Analysis Console View	44
3.2.9	Unresolved Symbols View	45
3.2.10	Preferences Dialog Box	46
	General Tab View	47
	Aggregate Tab View	48
	Tree Tab View	49
	Time Tab View	49
	Fragmentation Tab View (VxWorks Only)	50
	Database Tab View	51
3.2.11	Snapshots	53
	Taking a Snapshot	53
	Saving a Snapshot	54
	Viewing Snapshots From a Previously Saved File	54
	Viewing the Database File	55
3.3	Menus and Icons	56
	Menu Bar	56
	Icons	59
3.4	System Viewer Event Integration	59
	Automatic System Viewer Support	60
4	Using Memory Analyzer	63
4.1	Introduction	63
4.2	Finding Memory Leaks	63
4.3	Finding Memory Hogs	68
4.4	Advanced Topics	69
5	Troubleshooting	71
5.1	Introduction	71
5.2	Messages	72

	Target Errors	72
	Host Errors	73
5.3	General Troubleshooting Tips	74
	Issues With the Target	74
	Issues With the GUI	80
	General Tips	81
	Known Issues and Workarounds	82
A	Kernel Abstraction Layer (KAL)	87
A.1	Introduction	87
A.2	Basis for Need	87
A.3	Procedure	88
	Setup	88
	Wind River Linux 2.0 Targets	89
	Other Linux Targets	90
A.4	Known Issues and Workarounds	92
B	Event Dictionary	95
	System Viewer Events	95
C	Glossary	99
	Index	103

1

Introduction

1.1	Introduction	1
1.2	Architectural Summary	3
1.3	Features	6

1.1 Introduction

Wind River Memory Analyzer analyzes memory usage in a running real-time embedded program. It provides a live summary of each allocated block of memory in the system, which helps you detect problems, such as memory leaks, early in your development process.

Memory Analyzer is designed specifically to analyze C, C++, and assembly language programs only. For operating system and processor versions supported by this release, please refer to the release notes for your platform.



NOTE: This document contains background information and process descriptions only. Detailed help with user interface operations is available by pressing the help key for your host while running Memory Analyzer.

Memory Allocation Problems

Dynamic memory allocation is one of the most powerful tools available to software programmers. It is also one of the areas that is most error prone.

Memory problems can manifest themselves in many ways:

- Memory may be allocated but never properly deallocated — a memory leak.
- A function can write to an area of memory that belongs to another process — memory corruption.
- The available blocks are too small to be useful — memory is fragmented.

Memory use can waste processor cycles; for example, small allocations that are used and then freed can sometimes be replaced with a single allocation or static usage. Memory can run out because bursts of activity requiring many different parts of the software may need more memory than anticipated by the designers of each individual part.

Embedded systems must run for months or years unattended and without maintenance. If dynamic memory errors are not found and fixed in development, they are likely to cause an unacceptable failure in the deployed system.

Memory Analyzer is designed to help identify these mistakes and allow you to determine their cause. Memory Analyzer lets you do the following:

- Analyze production code without special compilation.
- View the call stack and process that allocated or freed each piece of memory.
- Identify potential memory leaks by viewing all unfreed allocations.
- Visually inspect memory for excessive use by a particular task, process, or thread (and, in VxWorks, for memory fragmentation).
- Detect memory usage patterns that should be optimized.

Memory Analyzer Overview

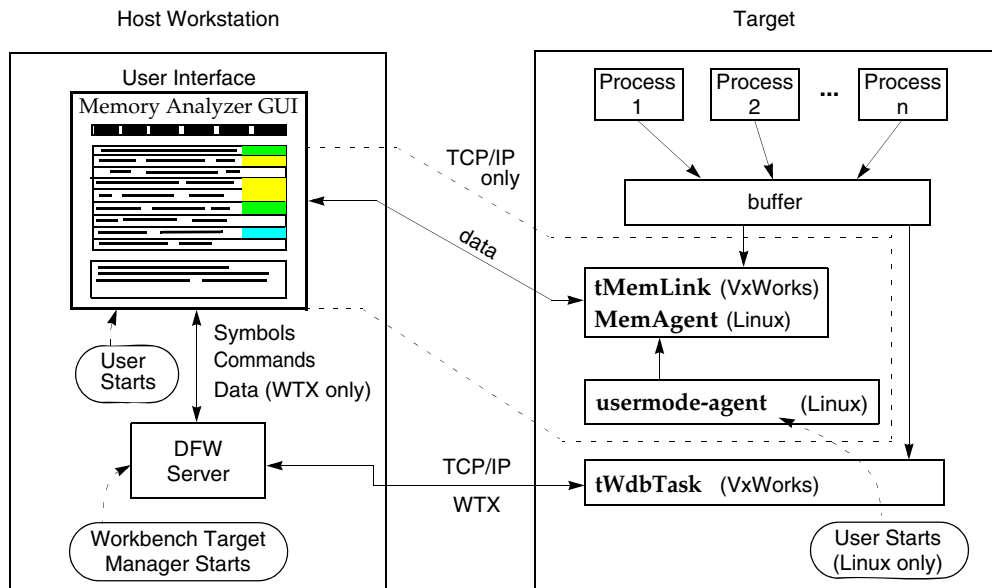
Memory Analyzer includes a collection agent, which runs on the target and patches the lowest-level memory allocation and deallocation functions to collect results and call stack data each time the functions are called. When memory is allocated (or freed), the patched function first creates an Allocation Record (AR). The patched routines then write the ARs into a message queue on the target. The Memory Analyzer graphical user interface (GUI), running on the host computer,

periodically retrieves messages from the AR message queue on the target by way of the communication link.

Because it patches the lowest-level routines, Memory Analyzer traces memory allocated by your code, by third-party libraries, and (for VxWorks) by the operating system. So in addition to your own allocation/deallocation routines, you can see, for instance, buffers allocated by your code, memory reserved for semaphores, and many other memory management functions.

1.2 Architectural Summary

The Memory Analyzer architecture consists of two main components: a collection agent that runs on a real-time VxWorks or Linux target, and a GUI that runs on the host, as shown in the illustration below.



There are some minor differences in the implementation of target architecture between VxWorks and Linux targets. These differences are outlined in the following sections.

VxWorks Targets

For VxWorks targets, the components communicate through the VxWorks Target Manager, including the DFW server, and if available, a TCP/IP link, or an optional WTX link.

Target-Side Modules

The target-side modules contain code required to intercept internal calls made to the lowest level memory allocation routines **memPartAlignedAllocInternal()** and **memPartFreeInternal()**, in the stack. The code then queues this data on the target for transmission to the host by way of the **tMemLink** routine if you are using a TCP/IP link, or **tWdbTask** routine for WTX. The target-side code is designed and implemented to take as little processing time as possible away from the running program.

Host-Side Interface

The host-side GUI runs on your host computer, and allows you to interact with the data collected by the target-side modules.

For loading and starting the host-side GUI, see the instructions in [2. Getting Started](#). For an explanation of each component of the GUI, see [3. The Memory Analyzer GUI](#), and an introduction on how to effectively use Memory Analyzer on a VxWorks target can be found in [4. Using Memory Analyzer](#).

Target-to-Host Link

The target and host have two primary methods of communication: TCP/IP mode and WTX mode, both discussed in [2.3 Starting Memory Analyzer](#), p.13.

Linux Targets

For Linux targets, the components communicate through the Workbench Target Manager, including the DFW server, over a TCP/IP link.

Target-Side Modules

The target-side modules contain code required to intercept calls made to the lowest level memory allocation and deallocation routines. The code then queues this data on the target for transmission to the host by way of the **usermode-agent** and **MemAgent** routines. Linux targets always communicate with the host over a TCP/IP connection. The target-side code is designed and implemented to take as little processing time as possible away from the running program.

Host-Side Interface

The host-side GUI runs on your host computer, and allows you to interact with the data collected by the target-side modules.

Target-to-Host Link

A Linux target only communicates with its host over a TCP/IP connection. This connection type is discussed in [2.3 Starting Memory Analyzer](#), p.13.

Wind River Linux 2.0 Targets

Run-Time Analysis Tools is integrated into the Wind River Linux 2.0 Build System as a root file system package. By default, it is part of the regular (non-**small**) root file system templates.



NOTE: Run-Time Analysis Tools can be added to a system configured to use a **small** root file system by including the template found in the **extra/scopetools** directory. Do this using either the **Workbench User Interface** or the **Command Line Interface** methods as outlined in [Wind River Linux 2.0 Targets](#), p.89.

Database Files

The Memory Analysis tool uses a set of files as a database to store all memory allocation and free event records. These files can consume a large amount of disk space depending on stack depth, duration of the run, and rates of malloc()'s and free()'s. The database files must not be on an NFS mounted disk drive. By default, the set of database files are stored in the temporary user directory. Also by default, the set of database files match the template "mem-targetserver-name-date-time.madb". This and other defaults mentioned below can be changed in the Database tab view of the Preferences dialog box (see [Database Tab View](#), p.51).



NOTE: To reduce the number and size of the Memory Analysis database files, you can uncheck **Append Timestamp** which will force the tool to delete and reuse the files named "mem-targetserver-name.madb". To reduce the number and size of the database files further, you can uncheck **Append Target Name** which will force the tool to delete and reuse the files named "mem.madb". However, you *must not uncheck* **Append Target Name** if you are planning on connecting Memory Analysis to multiple targets at the same time.

1.3 Features

Memory Analyzer embodies the following features:

- **Finds memory leaks**

You can see allocations that are not freed and may be memory leaks. These are displayed in the **Aggregate** view, covered in [3.2.2 Aggregate View](#), p.26.

- **Determines individual process memory usages**

Memory Analyzer dynamically shows you how much memory each process is using, and why. Process-based memory information can be found in the **Tree** View, described in [3.2.3 Tree View](#), p.31.

- **Displays dynamic memory allocations**

Watch memory allocations and frees graphically, as they occur, to understand the dynamics of your system memory usage. See how this works in the **Time** view, described in [3.2.4 Time View](#), p.34.

- **Identifies inefficiencies**

Quantifies heap usage by call stack (and for VxWorks only, by process, thread or partition), to identify memory hogs and inefficient use. Partition and call-stack heap usage can be most easily identified using the **Aggregate** view, while process usage can best be seen in either the **Aggregate** or **Tree view**.

- **View source code**

You can display the source code for the selected memory allocation/deallocation. For detailed information, see [3.2.7 Source Code Viewer](#), p.41.

- **System Viewer event integration** (VxWorks only)

Memory Analyzer is integrated with the Wind River System Viewer tool. When you launch Memory Analyzer, it automatically look for System Viewer support. If support exists, Memory Analyzer posts a System Viewer event for each sampled record, so you can see calls to traced functions directly within the System Viewer display. For detailed information, see [3.4 System Viewer Event Integration](#), p.59.

For a complete description of each of the GUI elements of Memory Analyzer, see [3. The Memory Analyzer GUI](#). The application of each of the above features is demonstrated firsthand in the development of solutions to typical problems presented in [4. Using Memory Analyzer](#).

2

Getting Started

- 2.1 Introduction 9
- 2.2 Requirements 10
- 2.3 Starting Memory Analyzer 13
- 2.4 Testing Your Installation 16
- 2.5 Usage Notes 19

2.1 Introduction

This chapter describes the process of installing, setting up, and running Memory Analyzer on either a VxWorks or Linux target platform. It gives you enough information to begin using Memory Analyzer to run a demonstration program supplied with the tool. For more information on using Workbench, see the *Wind River Workbench User's Guide*.

2.2 Requirements

Before you can run Memory Analyzer, you *must* first create a target connection in Workbench, then connect it to the target manager using the appropriate menu commands or icons in the **Remote Systems** view. For details on using the target manager, consult the *Wind River Workbench User's Guide: Target Manager View*, as well as your platform User's Guide. Instructions are given in this chapter for connecting Memory Analyzer to your target manager.

There are some dependencies Memory Analyzer places on your host operating system for resources that are specific to the target platform, summarized in the following sections.

VxWorks



NOTE: When running the target manager, the **-A** option *must* be present in the **Options** command line (in the target server **Properties** dialog box, in the **Target Server Options > Advanced Target Server Options** tab view). This forces all global and local symbols to be parsed and available for patching. Workbench does this by default, but if **-A** is absent for any reason, Memory Analyzer will not be able to find some symbols to be parsed, and you will not be able to collect and display data for those symbols.

- If you are using a PowerPC target, you should build your VxWorks Image Project with the extended vector addressing option enabled. Targets with large memories typically load Memory Analyzer at an address beyond the limited 26-bit PC-relative addressing normally used. This option is located in the **Components** tab view in the project **Kernel Configuration** view, under **operating system components, kernel components** in the tree. Right-click **Allow 32-bit branches to handlers** in the tree, then select **Include** to enable this option in your build.
- If your target board is running an **x86** processor, Memory Analyzer will run properly only if **frame pointers** are built into the code by the compiler. The compiler does this by default, but you must be aware that if you build your code with frame pointers turned off, you will encounter problems. For troubleshooting tips, see [Issues With the Target](#), p.74.
- If you want to collect data from real-time processes (**RTPs**), note that the RTP components in your kernel need to include **Shared Data Region** support. This support *must* be built into the kernel before running it by taking these steps.

- a. In the **Project Explorer** view, right-click **Kernel Configuration**, and select **Edit Kernel Configuration** from the list of options to open the **Component Configuration** view.
- b. Expand the components tree to **operating system components- > Real Time Process components**.
The **shared data region support in RTPs or kernel** option should be greyed out, indicating that support is not currently included in the kernel.
- c. Right-click the **shared data region support in RTPs or kernel** option to open a menu.
Note that properties for the field are displayed in the table below the tree. Note that the name is **INCLUDE_SHARED_DATA**.
- d. To open the **Include** wizard dialog box, select **Include** in the menu.
- e. In the **Include** wizard dialog box, check the **shared data region support in RTPs or kernel** check box, then select **Finish** to enable the support.
- f. Back in the **Project Explorer** view, right-click **Kernel Configuration** again and select the **Build** option to rebuild your kernel with the shared data region support you enabled.



CAUTION: If you are running an RTP on your target, the RTP spawn time limit *may* need to be set to 120 seconds or greater, and the back end request time limit *may* need to be set to 30 seconds. With the target disconnected, edit these values in the **Advanced target server options** group of the **Target Server Options** tab view in the target **Properties** dialog box.

If you do not attend to these items, the RTP initialization task may not receive sufficient CPU time to complete its execution before the RTP spawn time limit expires and causes the host to stop all tasks running in the RTP.

For more information, see *Wind River Workbench User's Guide: New Target Server Connections*, and also check your help key for **spawn time limit** while building the RTP task.



NOTE: Memory Analyzer supports RTPs on simulators, but with the following exception: it cannot report memory allocations happening inside the RTP itself, for example, if the RTP calls malloc directly. Memory Analyzer can report on RTPs that call general VxWorks system functions which happen to do their own memory allocation in the kernel for their own purposes.

- Memory Analyzer requires use of the **WDB** agent. The easiest way to ensure that your VxWorks Image Project (VIP) has WDB component support is to make sure one of the following kernel configuration Profiles is used in building your project:
 - NO_PROFILE
 - PROFILE_COMPATIBLE
 - PROFILE_DEVELOPMENT
 - PROFILE_ENHANCED_NET



NOTE: To help prevent target slowdowns, you *must* include the **INCLUDE_MODULE_MANAGER** loader component. Memory Analyzer communicates between target and host using the WDB target agent, so all default WDB components must also be included.

For more information, see *Wind River Workbench User's Guide, VxWorks Version: VxWorks Image Projects*.

- Wind River Run-Time Analysis Tools do not support connecting to a target using a **WDB_TIPC** connection. This means that if you are working in an **AMP** environment, you can only connect the Run-Time Analysis Tools to core 0 in AMP mode.

Linux

- Normally, users on Linux hosts will run Workbench as regular users, not root users. When Workbench is run by a regular user in a self-hosted setup (that is, running Workbench on your target machine), an attempt will be made to start the Run-Time Analysis Tools MemAgent program (or ProfileAgent on ProfileScope) as a regular user. MemAgent and ProfileAgent are both designed to be run successfully only by a root user. Therefore, in a self-hosted setup only, you will need to make the following changes to the MemAgent (and ProfileAgent if you are running ProfileScope) program file in order to run it as root user, even if you are using Workbench as a regular user:

```
$su
$cd /usr/scopetools-6.0
$chown root MemAgent ProfileAgent
$chmod +s MemAgent ProfileAgent
$exit
```

This procedure only needs to be done for self-hosted operation. For regular operation (separate host and target), this is all taken care of for you.

- Unless you are running self-hosted (as described above), you must logon as **root**, and start **usermode-agent** on your target. Therefore, the programs **insmod** and **rmmod** must be in your PATH because **usermode-agent** depends on them. They must also have execute permission (logging on as **root** gives this permission by default).
- In the process of building your target root file system, the binary files needed for the target you are using are copied to the directory

`/usr/scopetools-6.0`

If you should see a file in that directory with a name like the one formerly used to specify your specific architecture (that is, target type/platform/compiler, such as **ppc85xxGPP2.0gcc4.1.2**), it is an empty file and should be disregarded completely.

- You may need to rebuild the Run-Time Analysis Tools **KAL.ko** module before using the tools. For details, see [A. Kernel Abstraction Layer \(KAL\)](#). For the special case of Wind River Linux 2.0, see [Wind River Linux 2.0 Targets](#), p.89.
- For Linux targets, the programs **insmod** and **rmmod** must be in your user PATH because **MemAgent** depends on them. They must also have execute permission (logging on as **root** gives this permission by default).
- If you are running Memory Analyzer for the first time, compile the Kernel Abstraction Layer (KAL), described in [A. Kernel Abstraction Layer \(KAL\)](#), so Memory Analyzer can interact with your kernel correctly. If you skip this step, Memory Analyzer might crash your system. In the future, if you change and recompile your kernel, be sure to recompile KAL every time so that Memory Analyzer is always interacting with your kernel correctly.
- Under no conditions can more than one instance of Memory Analyzer be connected to the same Linux target simultaneously.

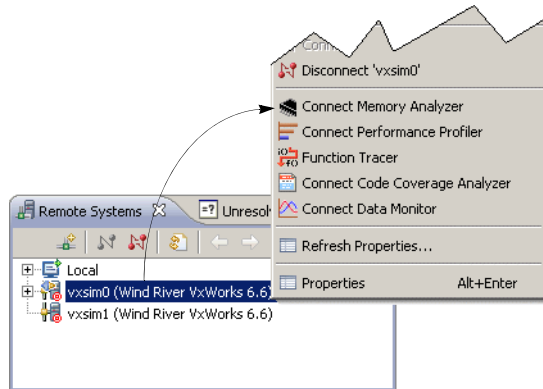
For more information, see *Wind River Workbench User's Guide, Linux Version: 6. Projects Overview*.

2.3 Starting Memory Analyzer

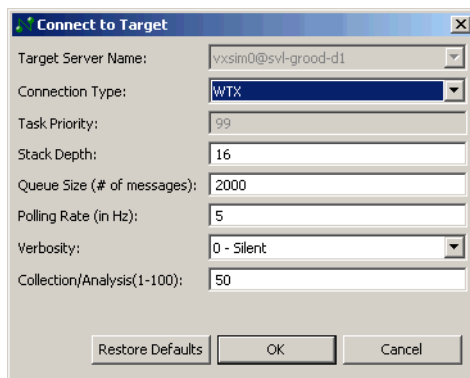
This section describes how to begin using Memory Analyzer in a real-time environment.

Initiating the Target Connection

The **Remote Systems** view contains the option for connecting Memory Analyzer to your target server. To connect, right-click the target server name and select the **Connect Memory Analyzer** command from the popup menu.



The **Connect to Target** dialog box opens, where you can select optional parameter values to be set when the Memory Analyzer GUI opens.



You can reset all the parameters to the Memory Analyzer default values with the **Restore Defaults** button. Click **OK** to begin the target connection process.

For information on setting parameters in the **Connect to Target** dialog box that opens, click in the parameter field and press the online help key for your host.



NOTE: The verbosity (number) you select in this dialog box maps to the verbosity (a character string choice) you can select in the **Analysis Console** view (see [3.2.8 Analysis Console View](#), p.44), according to the following mapping:

Connect to Target <--> Analysis Console

Verbosity 0 = **Severe, Warning, Info, and Config**

Verbosity 1 = **Debug**

Verbosity 2 = **Debug-hi**

Verbosity 3 = **Trace**

However, setting a verbosity level in one dialog box does not enter the corresponding value into the display in the other dialog box.

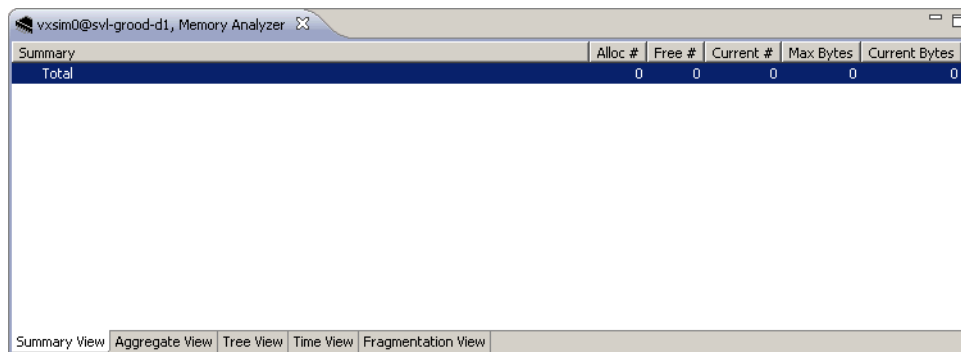


CAUTION: Setting target verbosity to a value greater than 0 may cause the MemAgent to needlessly generate an exceedingly large number of messages.

Generally, use the default value of 0 for verbosity, unless requested by Wind River Technical Support to help you diagnose a problem.

Opening the Memory Analyzer GUI

When you have connected to your target, the Memory Analyzer GUI opens automatically in the **Summary** view, an example of which is shown here.



For VxWorks targets, Memory Analyzer begins collecting and displaying data from all tasks and RTPs on the target. However, for Linux targets, Memory

Analyzer begins collecting and displaying data only after you have selected one or more processes from the **Process Selection** dialog box (see [Process Selection \(Linux Only\)](#), p.21). You can select the **Aggregate View** tab to see more details of the allocations that begin to display in the Summary view.

When you right-click anywhere in the view, a pop-up menu opens containing several items that are helpful in managing your data collection activities. Detailed help for each of these items is available using the online help key for your host.

2.4 Testing Your Installation

Having successfully started Memory Analyzer and connected it to your target, you can now begin to verify the installation and explore some of Memory Analyzer's capabilities. An easy way to get started is to try the following activities:

- Enter some shell commands.
- Run the demonstration program provided.

Viewing from the Shell

View results from Memory Analyzer as follows:

- In VxWorks, run a few commands in a shell to allocate some memory, and view the resulting memory usage statistics displayed in the Memory Analyzer GUI.
- In Linux, view the memory usage statistics in the Memory Analyzer GUI being gathered from basic system applications.

Running the Demonstration Program

For either a VxWorks or Linux platform, you can test your installation more thoroughly using the demonstration program `memscopedemo.c`, located in:

`WIND_SCOPETOOLS_BASE/target/src/vxworks/memscopedemo`

or,


`WIND_SCOPETOOLS_BASE/target/src/linux/memscopedemo_linux`

where `WIND_SCOPETOOLS_BASE` (an environment variable of the same name) is the root of the tree where you installed the Run-Time Analysis Tools.

In Workbench, perform the following steps to build and execute the demonstration program:

1. In the **Remote System** view, create a target connection with an appropriate name, if one does not already exist, then connect it to the target server.
2. Right-click the connection name and select **Connect Memory Analyzer**, then select **OK** in the **Connect to Target** dialog box to accept the default connection parameters.

Note that the Memory Analyzer opens in the **Summary** tab view.

3. Select the Memory Analyzer icon () on the Workbench toolbar to open the full Memory Analyzer perspective.

Verify that the status message in the **Analysis Console** view is:

Connected to target

If this message does not appear, check the Analysis Console view for error messages.

4. Build the Memory Analyzer example program `memscopedemo.c` following these instructions:
 - a. Right-click anywhere in the **Project Explorer** view and select **New**, then **Example** to open the **New Example** dialog box.
 - b. Select **VxWorks Downloadable Kernel Module Sample Project** in the **New Example** dialog box that opens, then click **Next**.
 - c. Select **The Memory Analyzer Demonstration Program** in the **New Project Sample** dialog box that opens, then click **Finish** to complete the project creation.

Notice that a new **memscopedemo** node now appears in the Project Explorer view. You now need to build the `memscopedemo.c` program.

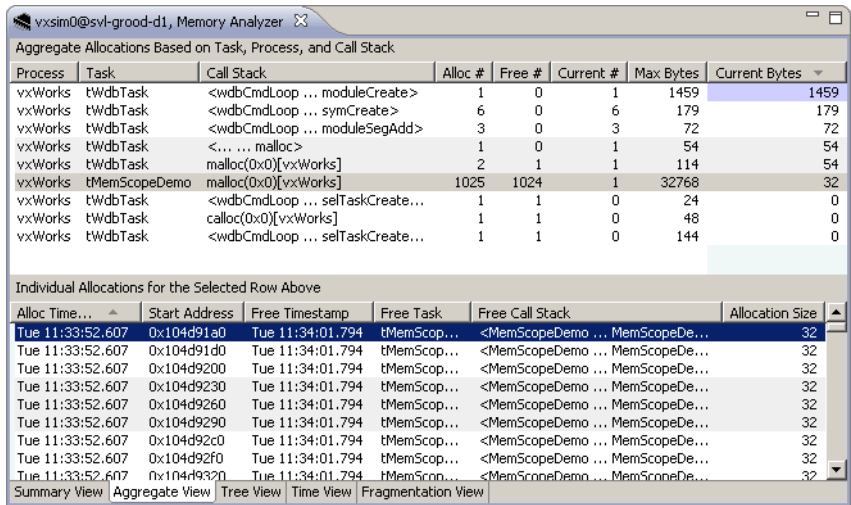
- d. In the Project Explorer view, expand the top (`memscopedemo`) node, then right-click the `memscopedemo (memscopedemo.out)` node and select **Rebuild Project** to build the binary files.

This program builds rather quickly, but you can follow the build progress in the **Build Console** view, as well as the progress meter in the **Build Projects** dialog box.

- 5. When the program has successfully built, execute it by following these steps.
 - a. In the Project Explorer view, right-click the **memscopedemo.out** node and select **Download**, then click **OK** in the **Download** dialog box that opens to download the executable files
 - b. In the Project Explorer view again, right-click the **memscopedemo.out** node and select **Run Kernel Task**.
 - c. In the **Run** dialog box that opens, in the **Kernel Task to Run** group, click **Browse** in the **Entry Point** field, and select **Downloads > memscopedemo.out > MemScopeDemo** as the binary files to be loaded, then click **Apply**.
 - d. Click **Run** to start the **MemScopeDemo** example program executing.

You will begin to see data being displayed in the GUI very shortly.

This demonstration program allocates several different blocks of varying size, and calls **free()** for some blocks but not for others. This figure shows, in the **Aggregate** view, an example of the output data generated by this demonstration program.



2.5 Usage Notes

The following subsections describe other factors that enter into consideration when using Memory Analyzer on a VxWorks or Linux target.

Symbol Resolution

For statistical analysis, and for performing symbol resolution, Memory Analyzer uses the ELF symbol table from the process object file. Without symbol resolution, all function names in the sampled data returned from the target would be displayed in the GUI simply as virtual memory addresses. For Memory Analyzer to print the corresponding meaningful function names, these addresses must be converted to their respective symbols. This is automatically done in the host-side GUI using DFW. For detailed information on this process, see the *Wind River Workbench User's Guide: Troubleshooting*.

You can view the list of symbols that Memory Analyzer has not been able to resolve, as well as the files in which they reside, in the **Unresolved Symbols** view, opened with the **Unresolved Symbols** tab (see [3.2.9 Unresolved Symbols View](#), p.45). If you experience any unresolved symbols in your analysis, refer to the *Object Path Mappings Page* section of the *Wind River Workbench User's Guide: New Target Server Connections* for helpful information.



NOTE: If the objects found on the host are not consistent with the objects being analyzed on the target (for example, they have been changed and recompiled), the symbol names may be skewed.

Symbol resolution is handled differently in the VxWorks and Linux versions of Memory Analyzer, as described in the following sections.

VxWorks Symbol Resolution

In VxWorks, Memory Analyzer uses the services of Workbench (specifically, the **dfwserver**) to resolve addresses into function names. Most of the time this arrangement successfully resolves all addresses into function names. However, in some situations, **dfwserver** may not be able to detect the presence of a new binary running on the target. For example, if a target has its own file system and an RTP started from the corresponding file system, and the target file system is not mirrored on the host on which the **dfwserver** is running, **dfwserver** cannot access the symbol table of the corresponding binary. Memory Analyzer will then not be able to resolve the addresses from the corresponding RTP into function names. In

this case you must specify the correct object path mappings in the target connection properties of the corresponding target in Workbench.

For more information, refer to the *Object Path Mappings Page* section of the *Wind River Workbench User's Guide: New Target Server Connections*.

Linux Symbol Resolution

For Linux targets, the GUI gathers profile data sent from the MemAgent on the target. Included in that process is looking up routine names that correspond to a list of addresses within each process on the target. These symbols are resolved by supplying the pathname(s) to object files for all code and library object files used by the target process.

Patching

Memory Analyzer collects memory allocation data from your target code by patching individual allocation/free routines that are ultimately responsible for manipulating blocks of memory. When one of those routines is invoked, a small, fast routine records the number of bytes being allocated or freed, as well as the sequence of instructions leading up to that invocation, into a target buffer. Memory Analyzer periodically transfers the contents of this buffer to the host, where it is analyzed and reported in the GUI.

Using memrun (Linux Only)

If you start your Linux target program and then use Memory Analyzer to select your target program for analysis, you could miss memory allocations that are happening between the time you start up your target program and enable Memory Analyzer to analyze your target program.

If it is important for you to not miss this early data analysis, you should use the **memrun** utility to start your target program.



NOTE: The **memrun** utility is only available for Linux targets.

memrun is an auxiliary program that performs the following steps for you automatically:

1. Loads the specified Linux target program along with its required libraries.

2. Patches all the routines in the target program to collect their data (so you do not have to open the Process Selection dialog box and manually select each process).
3. Waits for the GUI if it is not running yet, then starts the target program.

The result is that collected and analyzed data begin to be displayed in the GUI only after Memory Analyzer is up and running, so you will see all of your program memory allocations/frees from the beginning; nothing will be missed. You should use the **memrun** utility, especially if your target program only runs briefly, or if you are particularly interested in the earliest collected memory-operation data.

To use **memrun** to start your target program, enter the command

```
memrun target_program_name
```

where *target_program_name* is the name of your Linux target program. **memrun** detects if Memory Analyzer is running, and waits with a message for you to start it if it is not yet running.



NOTE: If you want to select additional processes for data gathering using the **Process Selection** dialog box, you can still do that even if you use **memrun** to start your target program.

Process Selection (Linux Only)

After starting Memory Analyzer with a Linux target, but before receiving memory usage statistics, you must select one or more active processes you would like to analyze. Do this using the **Process Selection** dialog box. This dialog box automatically opens when Memory Analyzer is initialized on a Linux target. Processes selected in this dialog box are patched in the target, and data from them is collected and analyzed by Memory Analyzer.



WARNING: If you see **MemAgent** in the **Process Selection** dialog box, *do not select it!* Doing so will result in unpredictable behavior, including possible system crash.



NOTE: For a Wind River Linux kernel, if you try to select a process in the **Process Selection** dialog box, and get a **Failed to patch** notice due to routines such as **kmalloc** and **vmalloc**, the reason for the error is that the process is a kernel thread. For Wind River Linux kernels, Memory Analyzer supports the analysis of user-space processes only. **Dynamic allocations that occur in a Wind River Linux kernel memory region cannot be analyzed by Memory Analyzer.**

For help on using this dialog box, press the online help key for your host.

Thread Analysis (Linux Only)

For Linux 2.4 target kernel analysis, threads are treated as regular processes. If, for example, you run a program containing 4 threads, you will see 4 entries for that program in the Process Selection list (see the figure above). Each entry will have the program name, followed by the thread's unique Process ID. If you select any one of the 4 threads to be analyzed by Memory Analyzer, all the threads in that program will be analyzed.

For Linux 2.6 target kernel analysis, threads are treated as lightweight processes that are grouped together under the process's initial Process ID. This means that in the Process Selection list there is only one entry for a multi-threaded process. Selecting that one entry to be analyzed causes all threads in that process to be analyzed.

3

The Memory Analyzer GUI

- 3.1 Introduction 23
- 3.2 The Memory Analyzer GUI 24
- 3.3 Menus and Icons 56
- 3.4 System Viewer Event Integration 59

3.1 Introduction

Memory Analyzer is a GUI-oriented application. It uses views, menus, and toolbar icons to allow you to see inside the target memory and determine what is going on. This chapter describes the individual Memory Analyzer views that display memory usage data. Each view provides a different tool to help you see and understand timing issues, call stack identities, and other complex interdependencies that point to the source of memory leaks. Data is displayed dynamically in all the views, as it is collected, and you can have multiple data views open at the same time.

3.2 The Memory Analyzer GUI

The Memory Analyzer GUI comprises the following views and dialog boxes common to both VxWorks and Linux platforms (except as noted):

- **Summary View**
Shows a summary of memory allocation and free data organized and listed by the tasks that performed them.
- **Aggregate View**
Provides a dynamic view of your target memory usage. This is the initial startup (main) view. For details, see [3.2.2 Aggregate View](#), p.26.
- **Tree View**
Displays each function in the call stack and its call progression leading up to the memory allocation, in a tree-like structure. For details, see [3.2.3 Tree View](#), p.31.
- **Time View**
Displays a time-line history of memory allocation and a corresponding table of data. For details, see [3.2.4 Time View](#), p.34.
- **Fragmentation View (VxWorks only)**
Displays a dynamically updating graphical map of VxWorks target memory allocation. For details, see [3.2.5 Fragmentation View \(VxWorks Only\)](#), p.36.
- **Details Viewport**
Shows complete call stack information for both the allocation and free operations for a row selected in any of the views above. For details, see [3.2.6 Details Viewport View](#), p.40.
- **View Source Code**
Displays the source code for a function selected in the **Tree View** or the **Details Viewport** view. For details, see [3.2.7 Source Code Viewer](#), p.41.
- **Analysis Console**
Displays error and warning messages that may be useful in determining connection or activation problems. For details, see [3.2.8 Analysis Console View](#), p.44.

- **Unresolved Symbols**

Displays the names of object files containing the symbols needed to resolve the unknown function names, currently displayed as hex numbers. For details, see [3.2.9 Unresolved Symbols View](#), p.45.

- **Preferences**

Provides a way to customize some appearance settings for the Memory Analyzer GUI. For details, see [3.2.10 Preferences Dialog Box](#), p.46.

- **Snapshot**

Creates a static view of your target memory usage in a new **Aggregate** view. For details, see [3.2.11 Snapshots](#), p.53.

- **Other**

Dialog boxes warn you of a variety of circumstances, such as connection problems, reminding you to initialize target libraries, and displaying target errors.

3.2.1 Summary View

The **Summary** view is initially opened when you start Memory Analyzer, but it can be reopened at any time using the Summary View tab.

Summary	Alloc #	Free #	Current #	Max Bytes	Current Bytes
Total	138	136	2	29064	28632
vxWorks	138	136	2	29064	28632
tJobTask(10397220)	0	2	-2	0	0
tWdbTask(1043cff0)	13	3	10	1872	1724
tShell0(104bdd88)	117	119	-2	26812	26812
Unknown(103d6b00)	8	12	-4	380	96

Expand

Summary View | Aggregate View | Tree View | Time View | Fragmentation View

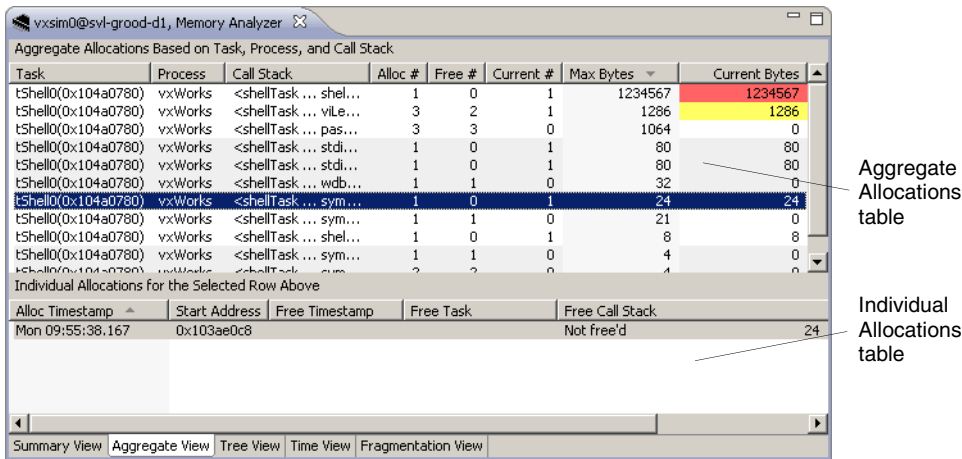
This view contains a single table with a tree-format list of tasks on your target that have allocated or freed memory. Each row in the table represents a function in

your source code. The content of the data columns is described in detail in [Aggregate Allocations Table](#) below.

When you right-click a row in the table, a pop-up menu opens with the single option **Expand**, which will expand the tree to expose all the branches below the one currently selected. In the expanded state, the button becomes **Collapse**, which reverts the selected branch, and all the branches below it, to the single starting node.

3.2.2 Aggregate View

Open the **Aggregate** view with the **Aggregate View** tab at the bottom of the Memory Analyzer view.



This view contains an **Aggregate Allocations** table in the upper half, and an **Individual Allocations** table in the lower half. If your target code is not yet running, the small amount of data displayed in the view reflects only the background tasks running on the target.

The user interface (menu items, toolbar icons, buttons, and popup menus, as well as data and column heading descriptions) are all described in context-sensitive online help by pressing the help key for your host. The following sub-sections describe in detail the major parts of the **Aggregate** view.

Aggregate Allocations Table

The **Aggregate Allocations** table in the upper half of the **Aggregate** view (see the figures above) dynamically displays the memory allocation statistics for all the functions and tasks that allocate memory in your target. (For Linux users, only the processes you selected in the **Process Selection** dialog box have their statistics displayed; see [Process Selection \(Linux Only\)](#), p.21.) This table enables you to locate the source of memory leaks and to identify functions or tasks that use large amounts of memory.



NOTE: As a reminder, in VxWorks, the program module currently being executed is referred to here and elsewhere in this user's guide as a *task* (or *thread*). For Linux users, *task* (or *thread*) should be replaced with the word *process* in every place where it means the equivalent of *task* (or *thread*).

The **Aggregate Allocations** table enables you to see the following:

- Functions that are allocating memory.
- How much memory they are allocating.
- How often they are allocating memory.
- Functions or tasks (if any) that are not freeing the memory they allocated.

Table Format

The table displays allocation statistics in columns within a row, with rows being added dynamically as memory allocation statistics are gathered.

Rows

Each row in the table is the sum of all memory allocation records (in the same partition, for VxWorks) that have the same task ID and function call stack signatures. Thus, if a given task allocates memory in exactly the same way, following the same call stack, the memory-allocation records are displayed in the same row of the table, but the number of counts and the number of allocated bytes are incremented.

Conversely, if memory is allocated by the same line of source code, but is called by a different task or through a different call stack, the allocation records are displayed in different rows in the display table.

When you select a record in this table, the allocation is highlighted, and the table in the lower half of the view displays the free details for all allocations that are

collected by the selected aggregate allocation above. This table is described in [Individual Allocations Table](#), p.30.

Columns

Each row contains several fields (columns) representing the statistical data collected for each call stack. The first three columns consist of the task/process ID that allocated the memory, and the call stack itself, formatted in a tree such that it can be expanded to show the full stack up to the maximum displayable depth. Finally the allocation/deallocation data is displayed.

Call Stack

The function call stack lists the nested function calls that led up to the allocation. The call stack helps you pinpoint memory calls in your code.

The call stack, as it appears in each row, is displayed by default in an abbreviated form as:

$name_1 \dots name_N$

where $name_1$ is the name of the first function below **Main** in the nested sequence of function calls leading up to the memory allocation, and, following the ellipsis, $name_N$ is the name of the last function in the sequence—the one that actually called the memory allocation routine. The following is an example of this call stack display.

```
<MemLinkEngine...RTilex>  
<MemScopeTrace...MemScopeDemo2>  
<wdbCmdLoop...taskCreate>
```

To distinguish between allocations arising from different modules that call the same function, turn on the display of module names by right-clicking in the selected row and selecting **Show Modules** (see description below). The following pattern shows how the call stack will now appear.

$name_1[module\ name_1] \dots name_N[module\ name_N]$

where $[module\ name_n]$ is the module containing the named function. The following example shows the previous call stack with this option turned on.

```
<MemLinkEngine[memscope.so]...RTilex[scopeutils.so]>  
<MemScopeTrace[memscope.so]...MemScopeDemo2[memscopedemo.so]>  
<wdbCmdLoop[vxWorks]...taskCreate[vxWorks]>
```

Likewise, you can distinguish between allocations arising from different offsets within the same function by right-clicking in the selected row again and selecting **Show Offsets** (see description below). The following pattern shows how the call stack will now appear.

$name_1(offset_1)[module\ name_1]...name_N(offset_N)[module\ name_N]$

where $(offset_n)$ is the offset, in bytes, into the current function at which the call to the next function occurred. The following example shows the call stack with both of the above options turned on.

```
<MemLinkEngine(0x138) [memscope.so]...RTI|ex(0x178) [scopeutils.so]>
<MemScopeTrace(0x0) [memscope.so]...MemScopeDemo2(0x40) [memscopedemo.s
o]>
<wdbCmdLoop(0xabc) [vxWorks]...taskCreate(0x2c8) [vxWorks]>
```

Right-click anywhere in the table and select either entry to toggle it on or off. The setting remains, even between sessions, until you toggle it again.

For VxWorks only, the limit on the number of levels in the call stack is determined by the **Stack depth** parameter in the **Connect to Target** dialog box. If the actual call stack is deeper than this limit, the last function is still the one that allocated memory, but the first function in the call stack is not the top-level caller in the actual call stack.

Data

Individual data items appearing in the row following the call stack consist of the statistical analysis results, including the number of allocated and freed bytes, as well as the maximum and current number of bytes allocated. The number of current bytes allocated by each row is visually emphasized by user-configurable background highlight colors that change with the increasing value of the outstanding allocations. This use of color helps you to spot the comparative seriousness of the memory offenders, as well as helping to spot potential memory leaks.

Customizing the Format

You can adjust and sort the different elements of the **Aggregate Allocations** table (and any other Memory Analyzer tables) in the following ways:

- To sort the rows by a selected column in descending order, select a column heading. To sort in ascending order, select the column heading again.
- To change the width of the columns in the table, drag the divider between the column headers.
- To help pinpoint possible memory problems, the background color of a **Current Bytes** column entry changes with its value at predetermined thresholds. These threshold colors and values are shown in [Table 3-1](#).

Table 3-1 **Default Color Coding for "Current Bytes" Values**

Color	Value Range
white	value < 1024
blue	1024 <= value < 4096
green	4096 <= value < 16384
yellow	16384 <= value < 65536
red	value >= 65536

The values and colors can each be modified using the color palette in the General tab view of the **Preferences** dialog box (see [3.2.10 Preferences Dialog Box](#), p.46). Help using this dialog box is available by pressing the help key for your host.

- Columns (statistics fields) can be added or removed using the **Aggregate** tab view in the **Preferences** dialog box (see [3.2.10 Preferences Dialog Box](#), p.46).

Individual Allocations Table

Below the Aggregate Allocations table is the **Individual Allocations** table, displaying information about allocated memory in a row selected in the Aggregate Allocations table that has been freed. This table is also formatted in rows and columns, but all the rows in this table are related only to the single row selected and highlighted in the Aggregate Allocations table (see [3.2.10 Preferences Dialog Box](#), p.46).

Rows

Each row in the Aggregate Allocations table can represent multiple allocations by the same task at different times (indicated by the **Allocs #** parameter), and each of those allocations can potentially be freed by a different call stack (task). The rows in the **Individual Allocations** table show each of those free operations separately. The number of rows in this table is equal to the number appearing in the **Allocs #** column of the selected row in the Aggregate Allocations table.

Columns

Each row contains fields associated with allocation and free operations selected in the Aggregate Allocations table. It includes the allocation and free timestamps, starting memory address of the allocation, task ID and call stack of the free task, and allocation size (in bytes). Help using these options is available by pressing the help key for your host.

The **Individual Allocations** table can be customized in the same way as the Aggregate Allocations table, as described in **Customizing the Format** (see [Aggregate Allocations Table](#), p.27).

Aggregate View Pop-Up Menus

When you right-click a row in the **Aggregate Allocations** or the **Individual Allocations** table, a pop-up menu opens with data display options.

Aggregate Allocations Table

Show Modules
Show Offsets
Show Outstanding Allocations Only
Clear Data

Individual Allocations Table

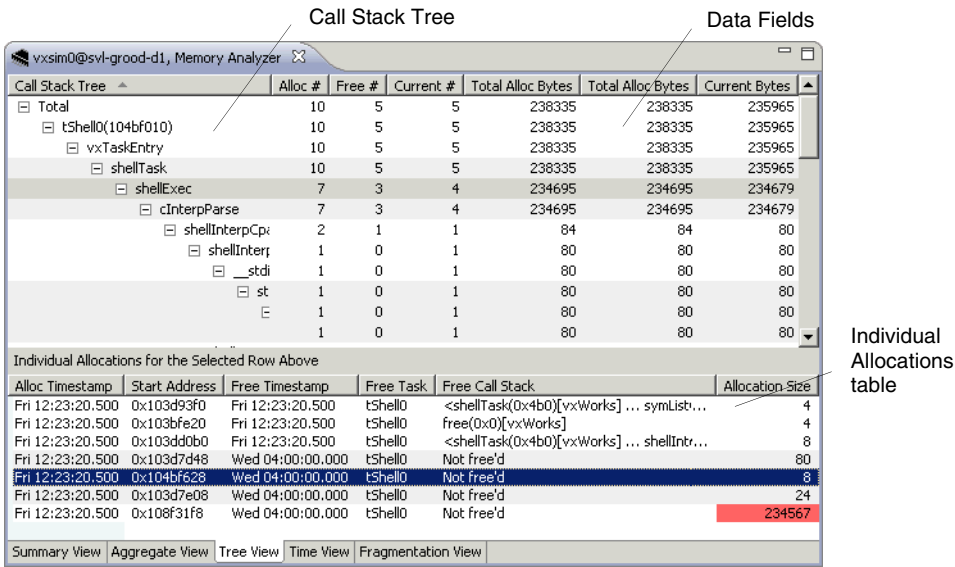
Show Modules
Show Offsets
Clear Data
Locate this record in Tree View
Locate this record in Time View
Locate this record in Fragmentation View

The menu in the **Individual Allocations** table includes the additional options **Locate this record in Tree View**, **Locate this record in Time View**, and **Locate this record in Fragmentation View**. Selecting any of these options opens the respective view, positioned and highlighted at the corresponding allocation/free row in that view. It thus allows you to correlate this data item with the same item in the other main views.

Additional help for this menu is available using the help key for your host.

3.2.3 Tree View

Open the **Tree** view by selecting the **Tree View** tab.



This view opens displaying a listing of the call stack trees for each function leading up to a memory allocation in its upper half. The **Call Stack Tree** entries are arranged in an expandable tree structure, and include the same data fields found in the Aggregate Allocations table of the Aggregate view (see [Aggregate Allocations Table](#), p.27).

In the lower half, the **Individual Allocations** table displays the free details for an entry selected from the Call Stack Tree in the Aggregate view above.

The following subsections describe in detail the major components of the Tree view.

Call Stack Tree

For each function in the **Call Stack Tree**, the nested functions below it can be exposed downward to any level until you reach the actual allocation call. At each function name in the tree for which there are yet more functions below, there is a "+" symbol that expands the tree one level below it by selecting it. The symbol then changes to "-" which, when selected, collapses (hides) everything that has been expanded below this function in the tree.

Individual Allocations Table

This table, in the lower half of the view, displays the free details for the selected tree entry in the allocations tree above. The details of this table are exactly the same as for the Individual Allocations table in the Aggregate view, described in [Individual Allocations Table](#), p.30.

The statistical data and column headings to the right of each function are the same as in the Aggregate Allocations table in the Aggregate view, described in [Aggregate Allocations Table](#), p.27.

The actual data columns displayed in this view can be selected using the **Tree** tab view in the **Preferences** dialog box, described in [Tree Tab View](#), p.49. The sequence in which the columns appear can be modified as described in [Aggregate Allocations Table](#), p.27.



NOTE: Color highlights in the Current Bytes column apply only to leaf nodes.

Tree View Pop-Up Menus

When you right-click a row in the **Call Stack Tree** or the **Individual Allocations** table, a pop-up menu opens with data display options.

Call Stack Tree

Expand Branch
View Source
Show Modules
Show Offsets
Clear Data

Individual Allocations Table

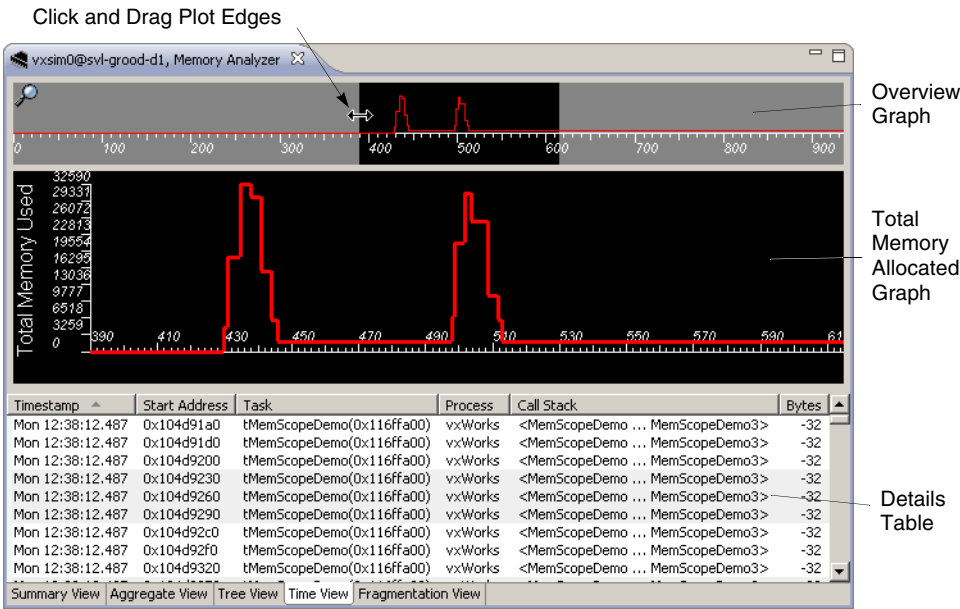
Show Modules
Show Offsets
Clear Data
Locate this record in Aggregate View
Locate this record in Time View
Locate this record in Fragmentation View

The menu in the **Individual Allocations** table includes the additional options **Locate this record in Aggregate View**, **Locate this record in Time View**, and **Locate this record in Fragmentation View**. Selecting any of these options opens the respective view, positioned and highlighted at the corresponding allocation/free row in that view. It thus allows you to correlate this data item with the same item in the other main views.

Additional help for this menu is available using the help key for your host.

3.2.4 Time View

Open the **Time** view by selecting the **Time View** tab at the bottom of the view.



In this window, collected data displayed in the Aggregate Allocations table (see [Aggregate Allocations Table](#), p.27) is translated into a chronological time-line display, showing the current total memory allocation for all allocations and frees executed since data collection began.

This section describes in detail the remaining major parts of the Time view and how it is used.

Graph Area

The Time view **Total Memory Allocated** graph appears in the middle area of the view. This is a dynamically updating graph of total memory allocated over time, initially plotting from the beginning of data collection to the present. However, you can zoom in (and out) on the data in this graph by simply clicking and dragging the outside edges of the **Overview** graph at the top of the page. The

information in the **Details** table below the graphs always relates to the data visible in the Total Memory Allocated graph.

The vertical (Y) axis plots total memory (in bytes) currently allocated. The Y value range (in bytes) increases to the greatest amount of memory that was allocated at any time since Memory Analyzer data collection began, regardless of the current total allocation.

The horizontal (X) axis plots time (in seconds), with start of data collection (0) on the left, and the total elapsed data collection time (in seconds) along the axis.

If you leave the graph in a zoomed-in position, it will remain stationary there, (but the Overview graph continues to scroll). To see new data plotted as it is being generated, drag the graph edges in the Overview graph back to the boundaries.

Details Table

This table, in the bottom half of the **Time View** window, shows details for each call stack that led to a memory allocation or free. The call stacks are displayed one per row, so there are two rows for a complete memory allocation/free sequence. If you select a row in the table, and the selected memory has been freed, a second row is also highlighted (for either the memory allocation or free, whichever is the opposite of what you selected).

Data in each row describe the allocations found in the Aggregate Allocations table (see [Aggregate Allocations Table](#), p.27). The row includes the target timestamp, the start address, the task and partition IDs, the call stack, and the number of bytes allocated. The number of bytes allocated by the line of code is displayed as a negative number if bytes are freed).

Help using these options is available by pressing the help key for your host. For detailed information about the call stack, see [Aggregate Allocations Table](#), p.27.

By default, all data columns are displayed in the table. You can, however, choose only the data columns you want to display in the **Details** table using the **Time** tab view in the **Preferences** dialog box, described in [3.2.4 Time View](#), p.34. The format of the columns as they appear in the table can be modified as described in [Aggregate Allocations Table](#), p.27.

More information on the **Call Stack** can be found at [Aggregate Allocations Table](#), p.27.

Time View Pop-Up Menu

When you right-click a row in the **Details** table, a pop-up menu opens with data display options.

Individual Allocations Table

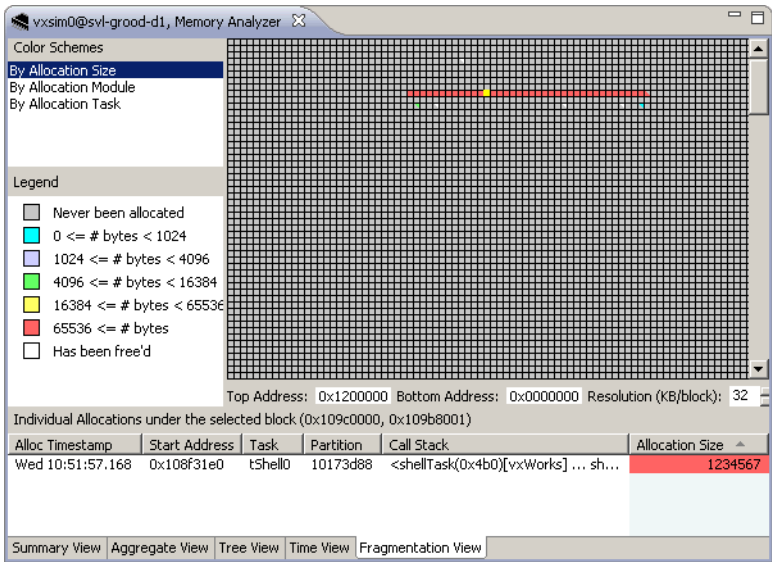
Show Modules
Show Offsets
Clear Data
Locate this record in Aggregate View
Locate this record in Tree View
Locate this record in Fragmentation View

This menu includes the additional options **Locate this record in Aggregate View**, **Locate this record in Tree View**, and **Locate this record in Fragmentation View**. Selecting any of these options opens the respective view, positioned and highlighted at the corresponding allocation/free row in that view. It thus allows you to correlate this data item with the same item in the other main views.

Additional help for this menu is available using the help key for your host.

3.2.5 Fragmentation View (VxWorks Only)

On a VxWorks target only, open the **Fragmentation View** with the **Fragmentation View** tab.



The **Fragmentation View** displays a graphical map representation of VxWorks target memory allocation since data collection began. This **fragmentation map** allows you to not only see the relative sizes of allocated blocks, but also view the address, size in bytes, and other allocation statistics for any selected block.

Fragmentation Map

This is a visual representation of memory as map blocks, each a square 5x5 pixels of a single color. Each map block represents a fixed number of contiguous bytes of memory. You can zoom in and out on the map by adjusting the **Resolution** arrows up or down at any time, or you can enter a resolution directly. The **Top Address** and **Bottom Address** fields display the current boundaries of the map. You can enter a byte value directly into either field to adjust the map display to locate and optimize your view.



NOTE: Any time you modify any of these three parameters, you must click **Enter** to complete the action and see the results.

Map Boundaries

The beginning of memory initially appears at the bottom right corner of the map, with the top left corner being the end (top) of memory. Newer allocations are always displayed starting at the top of memory, then to the right and lower in the view. Thus, each memory's start address will be higher than its end address. The fragmentation map initially shows all of memory. Any map block can be selected in the map with your cursor and its memory information details (if it is occupied) will be displayed in the **Individual Allocations** table below.

Color Schemes

The first (default) of three options, **By Allocation Size**, causes colored blocks in the map area to represent bytes of memory allocated by a single memory allocation module. The colors, shown in the **Legend** area of the view, show the correspondence between that color of mapped blocks and the size (range) of the allocation. The colors and corresponding size ranges are taken from the **General** tab view of the **Preferences** dialog box (see [General Tab View](#), p.47). You modify the default colors and size ranges in that dialog box, and the new values will remain, even across sessions, until you change them again. The default values can be reinstated with the **Restore Defaults** button.

Other Options

Of the two remaining options, **by Allocation Module** allows you to show the distribution of allocations made by up to the five modules making the highest total bytes of memory allocations (the "top five"). Note that here, too, the colors are as selected in the **General** tab view of the **Preferences** dialog box, but the byte size ranges in the **Legend** area are replaced by the module name(s). The resulting allocation block map will help you visualize the fragmentation created by those target contributors to memory usage.

The **by Allocation Task** option generates a memory block map similar to the by Allocation Module map described in the previous paragraph, but it displays results from the **tasks**, rather than modules, making up to the five highest memory byte allocations.

Data display/entry fields directly below the fragmentation map show these additional map parameters:

- **Top Address**

The address of the top-left (or end) of the allocated memory block. You can modify this value to include more or less of the available memory in the map.

- **Bottom Address**

The address of the bottom-right (or beginning) of the allocated memory block. You can also modify this value to include more or less memory in the map.

- **Resolution (KB/block)**

The number of bytes represented by one block (5x5 pixel square of one color) on the screen, selected using the arrows, with discrete resolutions marked in powers of two, from 1 to 32,768. This effectively zooms in and out in the map.

Individual Allocations Table

The **Individual Allocations under the selected block** table, in the bottom half of the window, shows details for each call stack that led to the memory allocation selected in the block map. The columns displayed in this table can be selected using the **Fragmentation** tab view in the **Preferences** dialog box (see [Fragmentation Tab View \(VxWorks Only\)](#), p.50).

The data items (selected as described above) are displayed from left to right in the columns of this table. They include the target timestamp, the allocation start address, the task and partition IDs, the call stack information, and the number of bytes allocated.

Help using these options is available by pressing the help key for your host. For detailed information about the call stack, see [Aggregate Allocations Table](#), p.27.

Fragmentation View Pop-Up Menu

When you right-click a row in the **Individual Allocations** table, a pop-up menu opens with data display options.

Individual Allocations Table

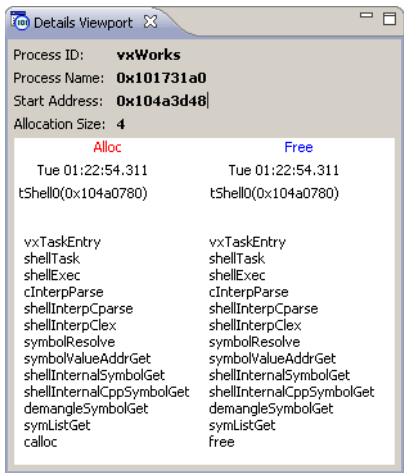
Show Modules
Show Offsets
Clear Data
Locate this record in Aggregate View
Locate this record in Tree View
Locate this record in Time View

This menu includes the additional options **Locate this record in Aggregate View**, **Locate this record in Tree View**, and **Locate this record in Time View**. Selecting any of these options opens the respective view, positioned and highlighted at the corresponding allocation/free row in that view. It thus allows you to correlate this data item with the same item in the other main views.

Additional help for this menu is available using the help key for your host.

3.2.6 Details Viewport View

The **Details Viewport** view displays complete information about a single allocation/free pair in one convenient place.



Each row in the Aggregate Allocations table can represent multiple memory allocations by the same task at different times. Each of those allocations can potentially be freed by a different call stack. The row displayed in the Individual Allocations table shows each of those discrete free operations for the row you selected in the Aggregate Allocations table. The **Details Viewport** view shows complete call stack information for both the allocation and free operations for the row selected in the Individual Allocations table of either an Aggregate view or a Tree view.

Double-clicking any entry in the table causes Memory Analyzer to display the source code for the selected routine (see [3.2.7 Source Code Viewer](#), p.41).

To distinguish between allocations from different parts of the same function, you can turn on the display of offsets by selecting the **Show Offsets** check box in the **General** tab view of the **Preferences** dialog box (see [General Tab View](#), p.47). You can likewise show module names by selecting the **Show Modules** check box in the **General** tab view.

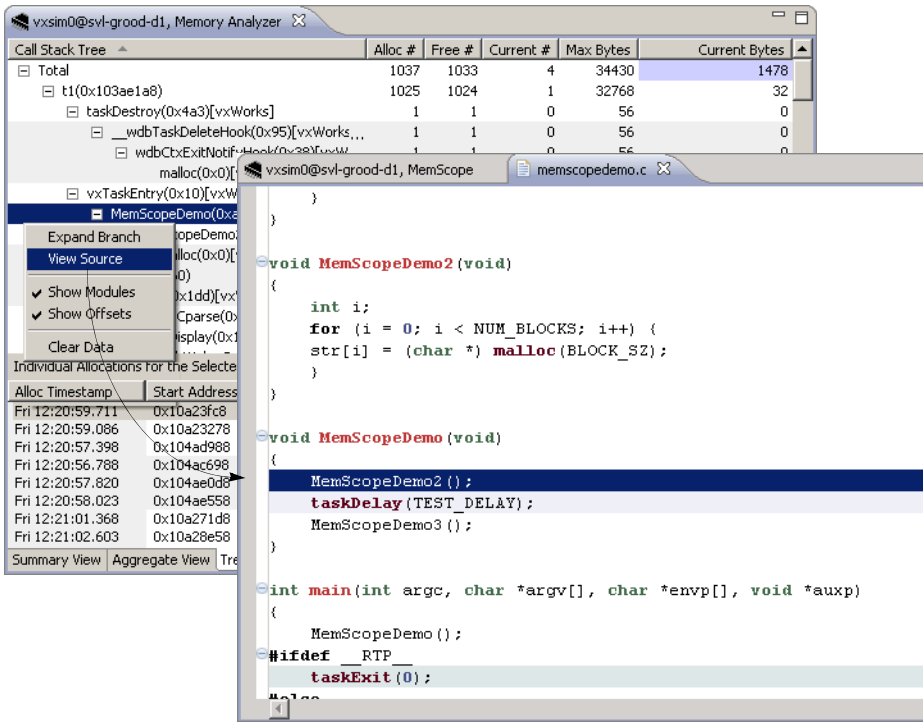
If you close the view, you can use the **Window > Show View > Details Viewport** menu command to open it again. Help with these view elements is available by pressing the help key for your host.

3.2.7 Source Code Viewer

You can view the source code containing a function displayed in the call stack tree of the **Tree View**, or an entry in the **Details Viewport** view. To do this, right-click the function and select the **View Source** option in the pop-up menu that opens. The source code containing that function will be displayed in another view that opens, sharing space in the Editor view along with the Memory Analyzer view.

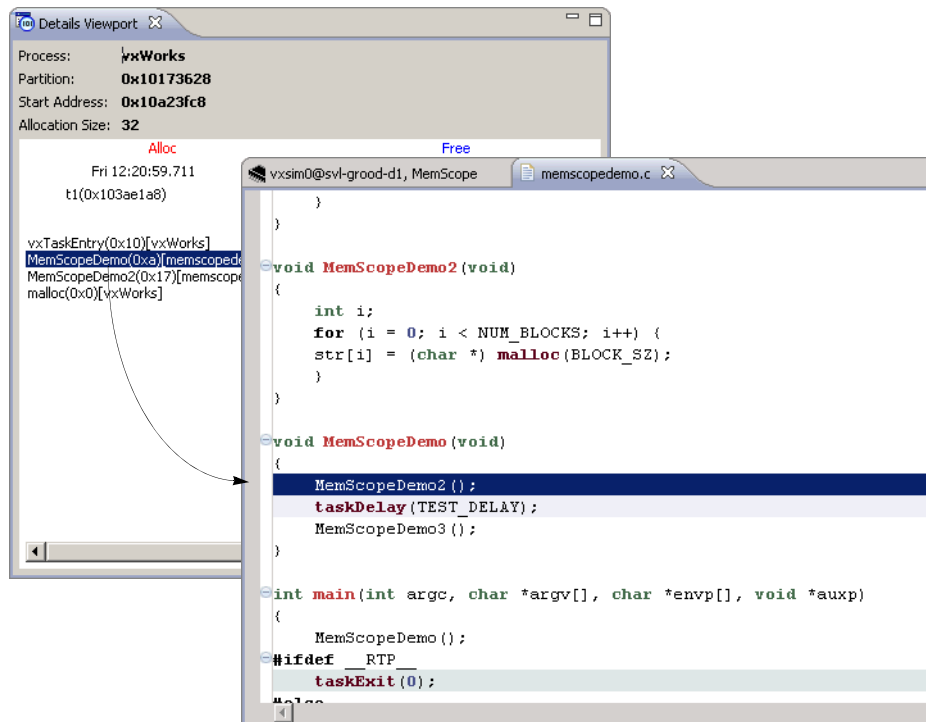
Call Stack Tree

To open a source code viewer in a Memory Analyzer view, right-click anywhere in any **Call Stack Tree** entry in the **Tree View** and click **View Source** in the popup menu that opens.



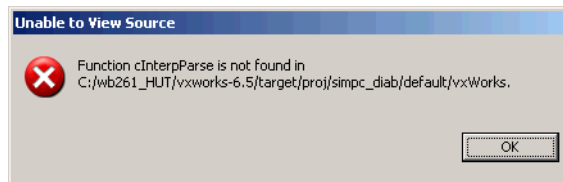
Details Viewport

In the **Details Viewport** view (see [3.2.6 Details Viewport View](#), p.40), double-click any call stack entry for your target code to open the source code view in the Memory Analyzer perspective.



In either of these methods, if the file can be found, the source code view opens with the entry point of the next routine shown in the call stack tree highlighted in the center of the page. You can navigate around in the view using the cursor.

In the event the source code file is not found (for instance, you moved your source code files since you last compiled them), a message to this effect is displayed in the **Unable to View Source** dialog box.



This could mean that the routine you have selected is a system routine, but if you know the routine is in your source code, you can fix this by selecting the **Window > Preferences > Run/Debug > Source Lookup** option. There you can configure other directory paths to search, click **OK**, then, back in the pop-up menu, select **View Source** again, as discussed above.

→ **NOTE:** If the source code has been moved from the location where it was compiled, you *must* add the current path to the search paths using the **Source Lookup** dialog box described above before Memory Analyzer can locate them.

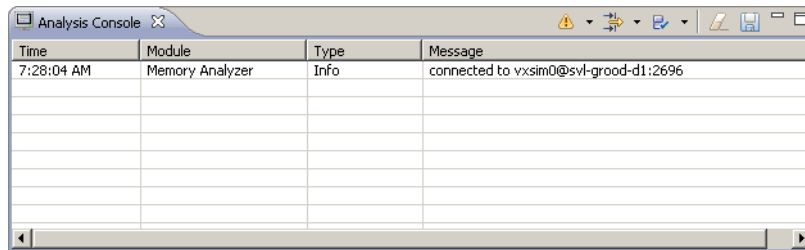
For more information on the Workbench **Source Lookup** option, see the *Wind River Workbench User's Guide: Launching Programs*.

→ **NOTE:** In order to use the View Source Code feature, you *must* have compiled your code with debugging information enabled. Because Memory Analyzer uses the DFW server, it works with whatever debug information standard is utilized by Workbench.

Recovering from these problems is also addressed in [Issues With the GUI](#), p.80.

3.2.8 Analysis Console View

Open the **Analysis Console** view with the **Analysis Console** tab.



This is where Memory Analyzer reports status, warning, and error messages generated by the host GUI. During normal operation (when **Verbosity** is set to 0), only a few messages are printed to the **Analysis Console** view. However, when you start Memory Analyzer with a non-zero target verbosity level, the amount of output can become significant. You should do this only when requested by Wind River Technical Support to help you debug any problems.

Optional activities include filtering messages, selecting columns to display, removing displayed messages, and saving messages to a file. Help with these view elements is available by pressing the help key for your host.

If you close this view, you can use the **Window > Show View > Analysis Console** Workbench menu command to open it again.



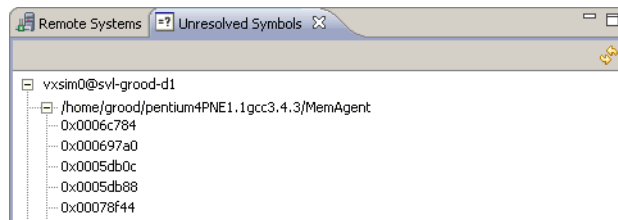
CAUTION: Setting target verbosity to a value greater than 0 may cause the MemAgent in the target (see [2.3 Starting Memory Analyzer](#), p. 13) to needlessly generate a large number of messages.

Generally, use the default value of 0 for verbosity, unless requested by Wind River Technical Support to help you diagnose a problem.

Help using this dialog box is available by pressing the help key for your host.

3.2.9 Unresolved Symbols View

Memory Analyzer gathers memory allocation data sent by the MemAgent on the target to the host. Part of that process includes looking up readable function names that correspond to a list of addresses within each process on the target. Unresolved symbols exist when the hexadecimal addresses representing function names in a task or thread cannot be resolved into meaningful function names by Memory Analyzer. When an address cannot be resolved into a function name, it appears in the **Unresolved Symbols** view along with the file in which it resides.



This view displays a tree-like structure consisting of pathnames and the hexadecimal numbers (symbols) at the end of each pathname that Memory Analyzer has been unable to resolve so far. They are grouped by the files in which they reside, and these are the files for which you must supply the full host pathnames using the **Window > Preferences > Run/Debug > Source Lookup** dialog box.

Since the view does not auto-refresh, use the **Refresh** icon (🔄) to refresh the screen with any additional symbols that were found to be unresolved since you opened the view (or last selected Refresh).

For a more complete discussion of symbol resolution, including the remedy for unresolved symbols, see [Symbol Resolution](#), p.19. Help using this view is available by pressing the help key for your host.

3.2.10 Preferences Dialog Box

The **Preferences** dialog box contains configuration parameters that modify the appearance of the GUI, change how Memory Analyzer collects profiling data, and how it analyzes that data. It is opened using the **Window > Preferences** menu command, then selecting **Wind River > Memory Analyzer**.

The dialog box contains the following tab views:

- **General**
Contains general, as well as call stack, display options, including a color palette for highlighting.
- **Aggregate View**
Contains parameters that modify the appearance of the **Aggregate** view.
- **Tree View**
Contains parameters that modify the appearance of the **Tree** view.
- **Time View**
Contains parameters that modify the appearance of the **Time** view.
- **Fragmentation View (VxWorks only)**
Contains parameters that modify the appearance of the **Fragmentation** view.
- **Database**
Allows you to select file parameters for the Memory Analyzer database file name, and to adjust the initial priority balance between host analysis and target data collection.

After making changes in any tab view, select **Apply** to save the changes and leave the dialog box open for more, or select **OK** to apply any changes and close the dialog box. Select **Cancel** to close the dialog box without saving any changes since you last selected **Apply**.

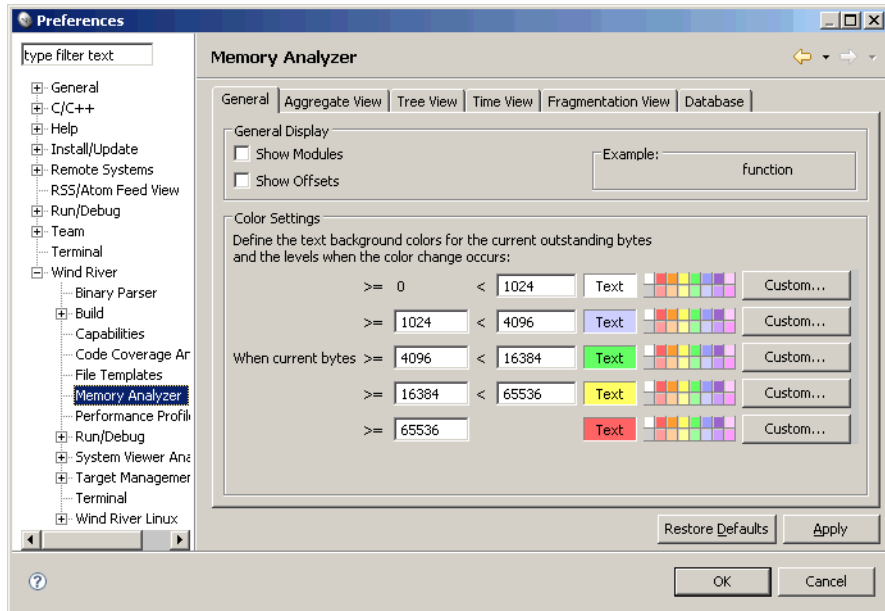
Memory Analyzer automatically saves the current settings of all configuration parameters to a registry file on a Windows host (or to a disk file on a UNIX host) when you exit the application. It automatically reloads that file on startup. You may also save configuration parameters to a specified file, or load them from a saved, file manually. For information on how to do this, see the chapter on Creating VxWorks (or Linux) projects in *Wind River Workbench User's Guide*.



NOTE: Any errors encountered in validating your selections will displayed at the top, just below the title bar, and will not allow you to proceed until corrected.

General Tab View

The **General** tab view contains parameters for selecting call stack display options (see [Aggregate Allocations Table](#), p.27), and a color palette for modifying the appearance of all the Memory Analyzer views.

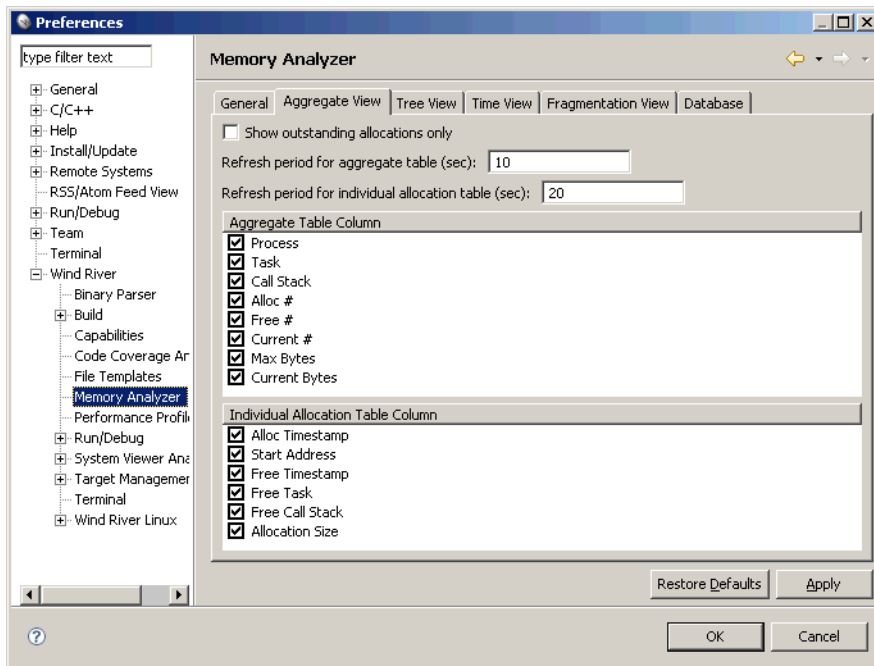


In this tab view you can select parameters that cause the certain tasks to be performed and colors to be used, including Show Modules, Show Offsets, and

Color Palette. Help using this tab view is available by pressing the help key for your host.

Aggregate Tab View

The **Aggregate** tab view contains parameters that modify the appearance of the **Aggregate** view (see [3.2.2 Aggregate View](#), p.26).

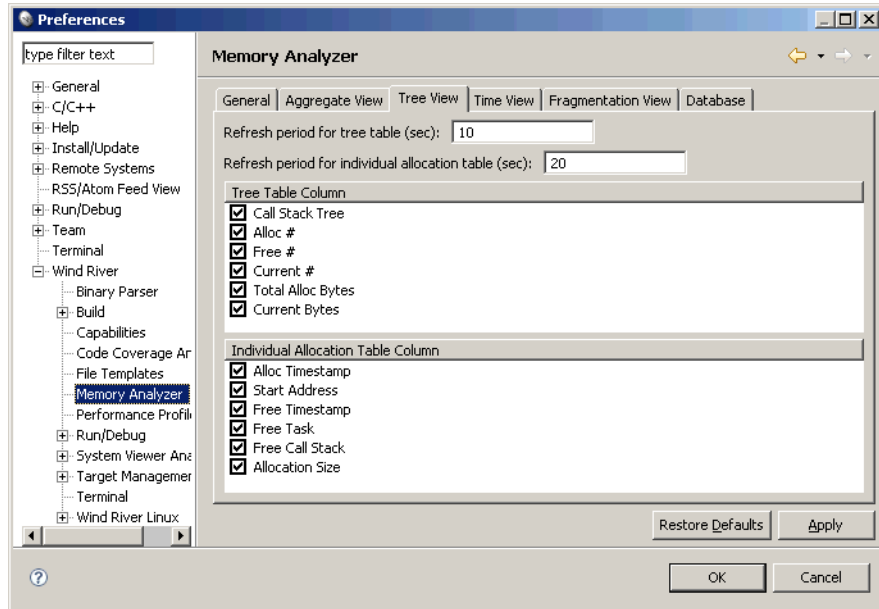


In this tab view you can select parameters that affect refresh rates and table columns to be displayed in the GUI. Help using this tab view is available by pressing the help key for your host.

Tree Tab View

The **Tree** tab view contains parameters, and two groups of check boxes, that modify the appearance of the **Tree** view (described in [3.2.3 Tree View](#), p.31).

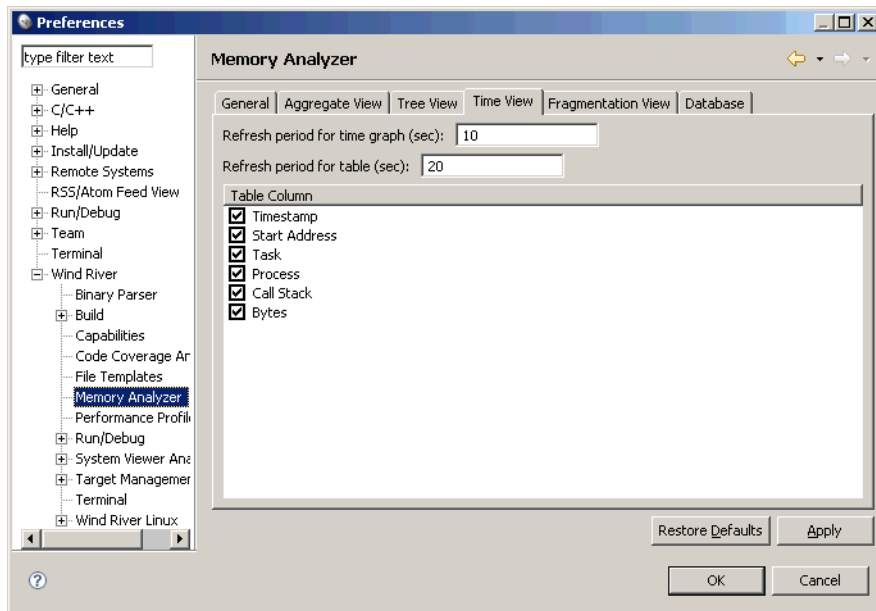
3



In this tab view you can select parameters that affect refresh rates and table columns to be displayed in the GUI. Help using this tab view is available by pressing the help key for your host.

Time Tab View

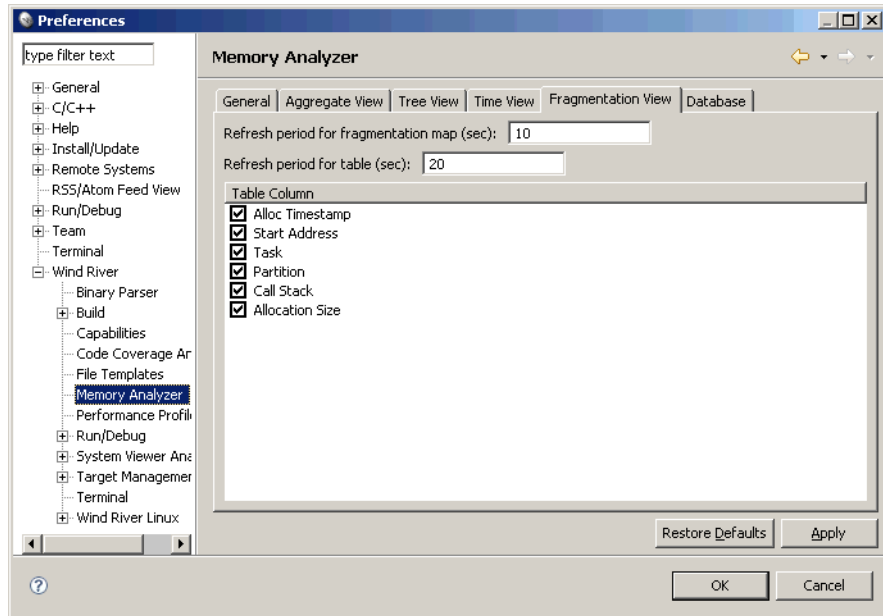
The **Time** tab view contains parameters, and a group of check boxes that modify the appearance of the **Time** view.



In this tab view you can select parameters that affect refresh rates and table columns to be displayed in the GUI. Help using this tab view is available by pressing the help key for your host.

Fragmentation Tab View (VxWorks Only)

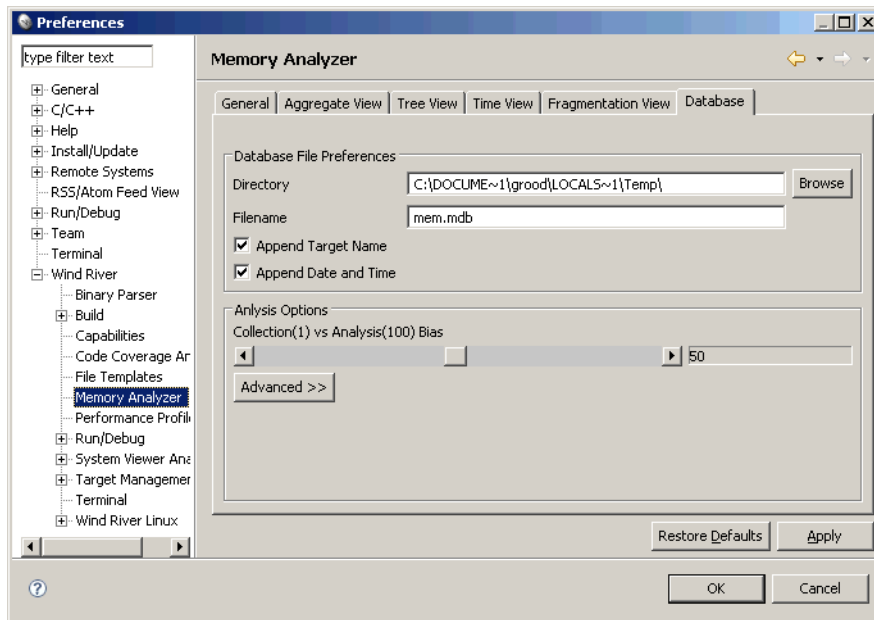
The **Fragmentation** tab view contains parameters, and a group of check boxes that modify the appearance of the Memory Analyzer **Time** view.



In this tab view you can select parameters that affect refresh rates and table columns to be displayed in the GUI. Help using this tab view is available by pressing the help key for your host.

Database Tab View

The **Database** tab view contains options for creating the Memory Analyzer database file described in [Viewing the Database File](#), p.55, and for setting the initial priority bias between analysis and data collection.



In this tab view you can select parameters that affect refresh rates and table columns to be displayed in the GUI.



CAUTION: Due to restrictions for file locking and consistency in the database engine, this directory *must not* be an NFS mounted directory.



NOTE: If both the **Append Target Name** and **Append Date and Time** boxes are checked, the Target Name option appears first, followed by the Date and Time, for example:

mem-walnut@svl-grood-d1-Aug05-112307.madb



CAUTION: Exposing the parameter values using the **Advanced** button overrides the slider values for the upper boundaries. You should *not* adjust these parameters directly unless requested by Wind River technical support to help in special cases. Use the Bias slider as described above.

Help using this view is available by pressing the help key for your host.

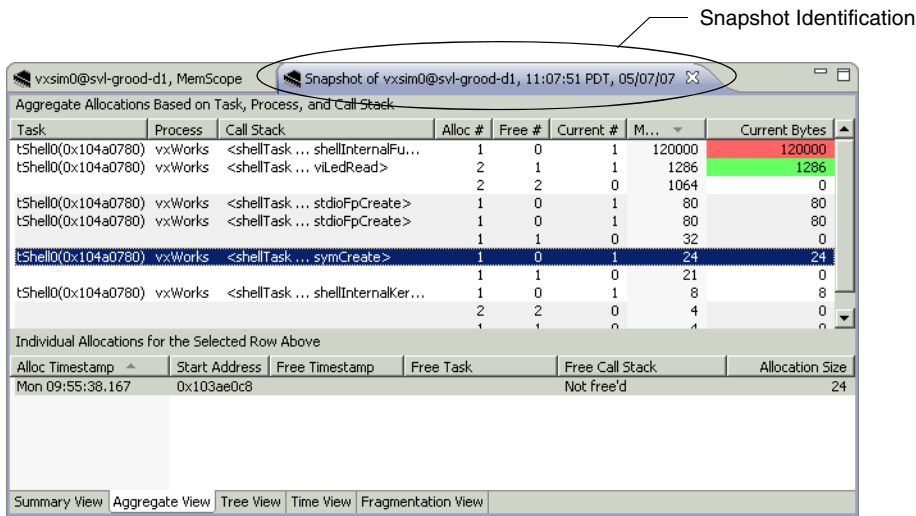
3.2.11 Snapshots

A **Snapshot** is a static view of the entire collection of memory usage statistics gathered up to the instant in this Memory Analyzer session that the snapshot is taken. Static means it can no longer be updated, even though data collection continues and is displayed in the live data views.

You can take a snapshot from any of the four main views described above using the **Memory Analyzer > Snapshot** menu command. This causes all the memory usage data collected from your target to be copied to a unique temporary file. It is initially displayed in a new **Aggregate** view, but you can then display it in any of the other views, just as with live data. The only thing you cannot do in a snapshot view is take another snapshot.

Taking a Snapshot

To take a snapshot, select the **Snapshot** icon on the Workbench toolbar. The captured data is then displayed in a new perspective view with a unique identifying time stamp tab (circled) at the top, as in this example.



The new perspective view is initially opened in the same tab view as the one in which the snapshot was taken, but you can display the snapshot data in any of the other views, the same as with live data.

Saving a Snapshot

Snapshot buffers are temporary, and will be lost when Memory Analyzer exits unless you save them. To save a snapshot buffer, follow these steps:

1. Select the **Save** icon to open the **Save** dialog box.
2. Enter a unique filename in the file-selection box.
3. Select **OK** to save the snapshot data to the file.

Viewing Snapshots From a Previously Saved File

To load and view a previously saved snapshot, follow these steps:

1. Select the **File > Open File** menu command.
2. In the **Open File** dialog box, navigate to the desired file and select **OK** to load it.

Memory Analyzer opens the selected file and displays its contents in a new **Aggregate** view. The time stamp identifies the snapshot by showing when it was taken.

Any number of snapshots may be open simultaneously, allowing you to perform comparisons of statistical data. Each snapshot view appears and operates nearly identically to its corresponding live view. Data collection and analysis continues when a snapshot is taken, but new data cannot be added to the snapshot once the snapshot has been taken. The only thing you cannot do in a snapshot view is take another snapshot.



NOTE: Even though the snapshot is always initially displayed in the **Aggregate** view, it contains all the data collected up to the time of taking the snapshot. Therefore, the snapshot data can be displayed in any view or mode, exactly as with the original data.

Viewing the Database File

As data is being collected and analyzed by the Memory Analyzer GUI, it is also being stored as a streaming data file of memory allocation statistical data from the target, on a persistent storage medium. Like a snapshot, this default streaming data file can be opened and viewed on the GUI using the **File > Open File** menu command, even while the data continues to be generated and stored in the file.

The unique feature of this file is that it is created, and is being saved, on a storage disk, in real-time as the data is being generated, rather than keeping the generated data in memory until it is later saved out to a disk file. If either the target or GUI system should fail at any point, all the data collected and analyzed up to that point is saved on disk for replay and continuing analysis.

The file can be opened for display in the Memory Analyzer GUI (similar to a snapshot), or copied to another file to be saved and replayed later in the GUI. This file has a default structure composed of the following elements:

mem-target name-timestamp.madb

where,

target name is the name of the target from which the data is being derived

timestamp is a timestamp constructed of these elements:

month/day-hhmmss.madb

where,

month/day is the current date, formatted, for example, as **Aug05**

hhmmss is the time, formatted, for example, as **114230**

These numbers represent the date and time when the target program was started, and serve to create a unique filename without any user intervention. You can, however, substitute your own filename, formatted in any desired manner, but it must have the **.madb** extension in order to be recognized as the streaming data file.

This data file will be terminated with EOF whenever Memory Analyzer is either deliberately shut down, or otherwise experiences a failure. When Memory Analyzer is invoked again, a new streaming data file will be opened, as described above. A streaming data file can be opened and reviewed at any time by simply using the **File > Open File** menu command.

3.3 Menus and Icons

The individual views that comprise Memory Analyzer are:

- **Memory Analyzer** main view
- **Details Viewport** view
- **Unresolved Symbols** view
- **Analysis Console** view

There are a number of menu items in the Workbench menu bar that pertain to these Memory Analyzer views, at least in part if not entirely. In addition, some of the Memory Analyzer views contain icons for actions applicable to that view. These items are described in the sections that follow.

Menu Bar

The functionality specific to Memory Analyzer is accessible from the following Workbench menu items:

- **Memory Analyzer**
- **Window**

Memory Analyzer Menu Item

This Workbench menu item appears when Memory Analyzer is started, and it contains the commands related to Memory Analyzer as described here.



The commands in the **Analyze** menu include:

- **Export**

Dumps raw memory allocation data to a text formatted file.

Opens a browser window where you enter or select a memory allocation data file to be saved in tab-delimited ASCII format, that can be used for further analysis such as in a spreadsheet application. After a header, each record

contains the task ID, memory address, size, and the call stack of the function that allocated the block.



NOTE: Exported data cannot be re-loaded into Memory Analyzer; use **File > Save** for that purpose.

- **Snapshot**

Creates a static view of your target memory usage in a new **Aggregate** view. For details, see [3.2.11 Snapshots](#), p.53.

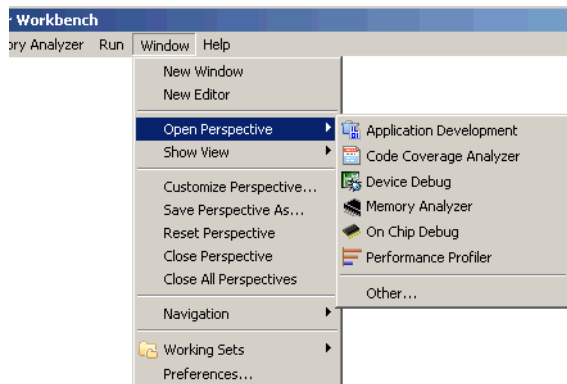
Window Menu Item

This Workbench menu item contains commands to open different perspectives, well as commands to open specific views. The **Window** commands applicable to Memory Analyzer include:

- **Open Perspective**
- **Show View**
- **Preferences**

Each of these **Window** commands is expanded and described separately below.

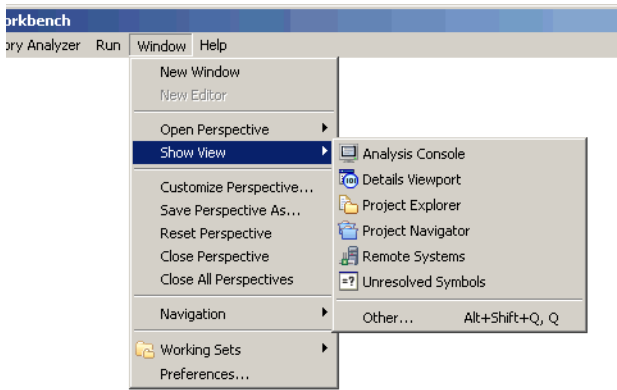
Open Perspective Menu Item



The commands in the **Window > Open Perspective** menu item applicable to Memory Analyzer include:

- **Memory Analyzer**
Opens the Memory Analyzer perspective, which will be the initial list of views previously shown in [3.3 Menus and Icons](#), p.56 above, or the specific views you had open when you last ran Memory Analyzer.

Show View Menu Item



The commands in the **Window > Show View** menu item applicable to Memory Analyzer include:

- **Analysis Console**
Displays system and error messages generated while running Memory Analyzer. If the verbosity level is non-zero, the **Analysis Console** window also displays debug messages. There may also be error messages from the communication connection with the target. It is described in detail in [3.2.8 Analysis Console View](#), p.44
- **Details Viewport**
This command opens the **Details Viewport** view. For the row currently selected in the **Individual Allocations** table of the **Aggregate View**, it shows allocation and free call stacks contents. It is described in detail in [3.2.6 Details Viewport View](#), p.40.

- **Unresolved Symbols**

Lists all the symbols Memory Analyzer has been unable to resolve into meaningful function names so far. It is described in detail in [3.2.9 Unresolved Symbols View](#), p.45. Unresolved symbols are discussed more thoroughly in [Symbol Resolution](#), p.19.

Preferences Menu Item

The **Window > Preferences** menu item, appearing at the bottom of the **Window** list, opens the **Preferences** dialog box. It is described in detail in [3.2.10 Preferences Dialog Box](#), p.46.

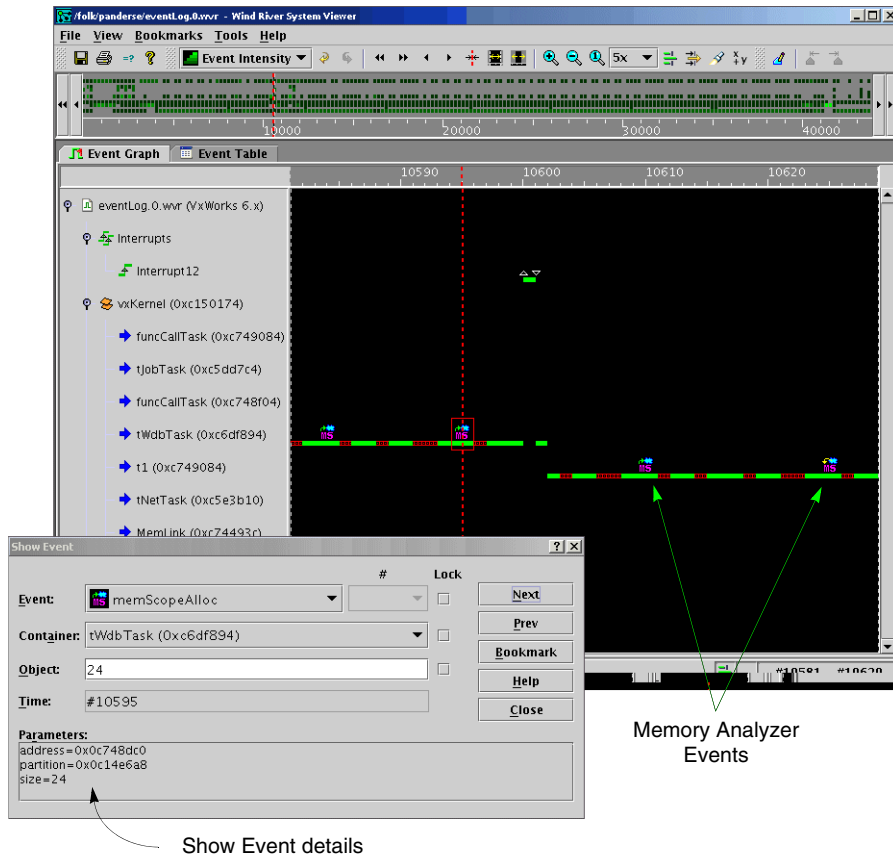
All menu items are also described in context-sensitive online help. For specific information, select a menu item in Workbench and press the help key for your host.

Icons

The description of icons in the various Memory Analyzer views can be found by selecting the icon and pressing the help key for your host.

3.4 System Viewer Event Integration

Memory Analyzer is integrated with the Wind River System Viewer to allow you to view memory allocation and free events directly in a **System Viewer** window.



You can see the memory allocation and free events alongside other operating-system events, such as task switches and semaphore calls.

For details on initializing and running the Wind River System Viewer, consult the *Wind River System Viewer User's Guide*.

Automatic System Viewer Support

When you load Memory Analyzer automatically from Workbench (see [2.3 Starting Memory Analyzer](#), p.13), the Memory Analyzer Setup code detects whether your target kernel supports System Viewer. If so, the Memory Analyzer target libraries

are initialized with System Viewer support. With this support enabled, Memory Analyzer will post a System Viewer event for every call stack trace record. Memory allocation and free events appear as two different types of events.

The description of events and triggers used by Memory Analyzer are found in Appendix [B. Event Dictionary](#). In analyzing your results, use the sequence numbers in the System Viewer events and in the Memory Analyzer **Aggregate Allocations** and **Individual Allocations** tables to match events with allocation and free records.

4

Using Memory Analyzer

- 4.1 Introduction 63
- 4.2 Finding Memory Leaks 63
- 4.3 Finding Memory Hogs 68
- 4.4 Advanced Topics 69

4.1 Introduction

This chapter contains practical examples of the kinds of problem scenarios you can solve with Memory Analyzer, using a step-by-step instruction format. It describes typical problems encountered in the development of optimized software for VxWorks and Linux operating systems, and how you can use Memory Analyzer to analyze and diagnose these problems.

4.2 Finding Memory Leaks

Memory leaks occur when allocated memory is not freed and is no longer needed. While each incident may only leak a small amount of memory, the total amount

over time may become very large. For most embedded systems, this is an unacceptable situation; eventually, the lack of memory will cause the system to stop functioning.

Once discovered, memory leaks are notoriously difficult to track down and eliminate. Painstaking examination of the source code may reveal some problems, but it is also common that the problem stems from misuse of a third-party library, for which the source code is not available. Memory Analyzer provides a more practical solution.

Because Memory Analyzer tracks every allocation and deallocation taking place in a task, it can discover memory leaks or increase confidence that the system is leak-free. If Memory Analyzer discovers memory leaks, instantaneous correlation of the leaked memory with the exact function call and call stack that allocated it allows the source of the leak to be identified immediately.

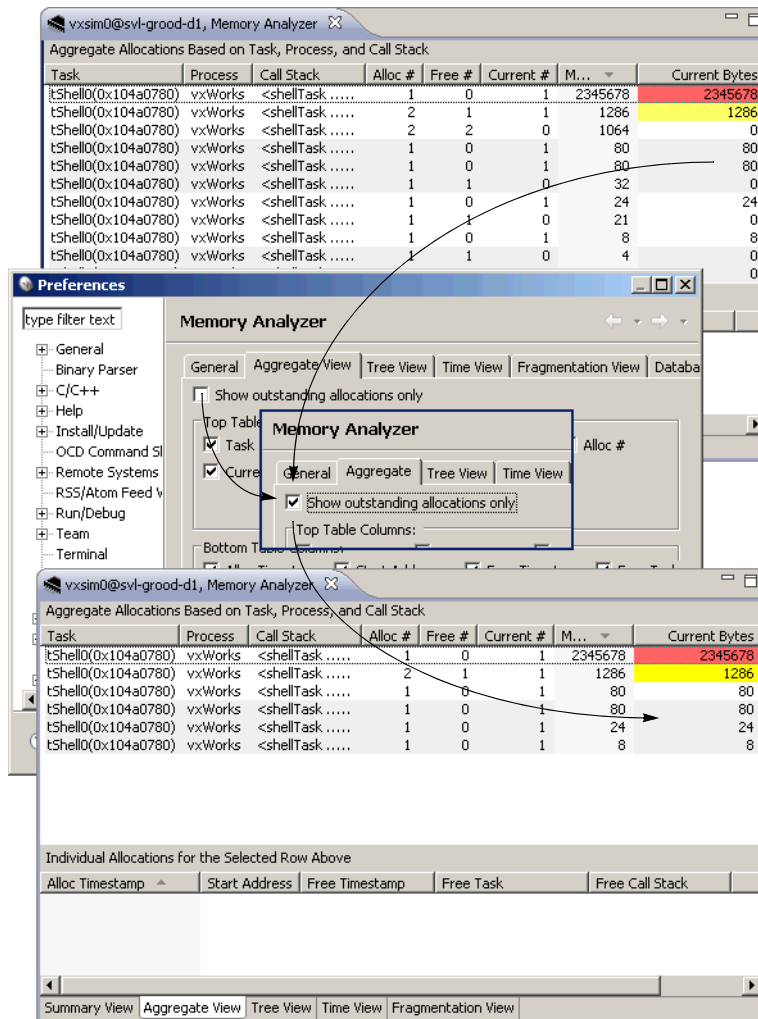
To detect leaks, use the following steps:

1. **Setup**
Start Memory Analyzer; initialize and start user application(s).
2. **Test**
Exercise the system.
3. **Exit**
Stop user applications (on systems where this is feasible).
4. **Analyze**
Determine if outstanding allocations are leaks.
5. **Iterate**
Repeat from step 1 or 2, if desired.

These steps are described in greater detail in the following paragraphs.

Setup

After starting Memory Analyzer and initializing and starting user code, data begins to be displayed in the **Aggregate** view, and may quickly fill the screen. To reduce the amount of displayed data when searching for memory leaks, you can instruct Memory Analyzer to hide all allocation rows that have been freed. Do this using the **Show outstanding allocations only** check box in the **Aggregate** tab view of the **Preferences** dialog box, opened with the **Windows Preferences** menu command, as in the following example.



NOTE: Memory Analyzer only tracks memory allocated after Memory Analyzer was started. When detecting leaks, Memory Analyzer should be started as early as is feasible in the bring-up cycle. For most projects, it is sufficient to start Memory Analyzer after the system has booted but before any user code is initialized and started. If your project contains custom code within the boot phase of the system, you may want to see [4.4 Advanced Topics](#), p.69.

Test

Because Memory Analyzer detects leaks that occur while the program is running, it is important to exercise as many areas of the program as possible after starting Memory Analyzer. Any available automatic test suites should be run in addition to any available QA tests or other test material. The greater the coverage of code and execution paths achieved by the tests, the greater the probability of exposing leaks.

If feasible, the current Memory Analyzer data set should be analyzed after every step of the testing process. This ensures that all allocations are understood and their correctness is verified. In larger projects this may be difficult, but Memory Analyzer can show the module related to each allocation if **Show Modules** is selected, which may help locate the module, or group of modules, responsible if an outstanding allocation needs to be understood.

You can add more allocations to the list by running different tests on the example. After running these tests, you can sort the display by the **Current #** column by selecting its column heading. This brings allocations that are currently outstanding to the top of the table. At this time, any of the outstanding allocation entries are candidates for memory leaks. Each one must be investigated and understood to assure that no leaks are taking place.

Exit

If the application is designed to be cleanly terminated, it is easier to discover whether an outstanding allocation is truly a leak; any intentionally unfreed memory should be freed when the application exits. However, on many systems, the application is not designed to quit. On these systems, the **Exit** step is reached when the system is no longer allocating and freeing memory in response to system tests.

Some systems may continually allocate and free memory; these transient allocations should be ignored at this point (unless they seem to be always increasing in **Current #**). Users with systems that do not quit may also want to iterate, as described in [Iterate](#), p.67, to better understand which allocations are expected and which allocations are leaks.

Analyze

Look at all outstanding allocations in the **Aggregate** view to determine if they are leaks. In addition to the call stack, the following information is available from the table entry in the **Aggregate** view to help understand why this memory was allocated:

- Task name and **id**.
- Process name.
- Number of times this allocation has been seen (**Allocs #**).
- Number of allocations from this entry that have already been freed (**Frees #**).
- Number of current outstanding allocations (**Current #**).
- Maximum bytes ever allocated at one time by this entry (**Max Bytes**).
- Number of bytes allocated currently by this entry (**Current Bytes**).

For more information about the **Aggregate** view, see [3.2.2 Aggregate View](#), p.26.

At this point, some knowledge of the intended behavior of the system becomes indispensable in diagnosing the seriousness of a potential memory leak. Once you have become familiar with the way Memory Analyzer depicts data, you can sort large quantities of allocations to find the ones that need attention.

Iterate

It may be desirable at this point to repeat the tests. Several options are available within Memory Analyzer for comparing the results of iterative tests:

1. Save the current data file. Open it using **File > Open** to view it alongside newly collected data.
2. Use the **Snapshot** facility to open a snapshot of the current counts.
3. Save the current data file using **File > Save**. Quit Memory Analyzer, then restart it. This completely clears out all data, instead of simply zeroing the counts.

After using these methods of clearing out the current data, either repeat from a clean system boot or repeat the tests from the current, quiescent state. Then place the results from the runs next to each other, and examine to see if the data is comparable. Potential leaks that appear in one test run but do not appear in another warrant closer inspection.

4.3 Finding Memory Hogs

Memory hog detection may help identify the programming team responsible for a module or task that is consuming too much memory. It may also be used to verify that everyone is staying within the limits established at design time.

Tasks That Hog Memory

You can observe Memory usage sorted by task in the **Tree** view, where every entry directly below the root node (**Total**) is a task. Use these entries to see how much memory is being used by each task.

Functions That Hog Memory

Detect functions that hog memory by using the **Aggregate** view and sorting on either the **Max Bytes** column or the **Current Bytes** column, depending on which is most interesting. This causes memory hogs to rise to the top.

Inefficient Memory Allocation Patterns

Using Memory Analyzer to trace memory allocation and deallocation patterns can help to show up inefficiencies that, when optimized, could reduce overall memory usage requirements.

Accounting for Peak Usage

Some systems need to be optimized to handle the common case, but must also handle the peak or burst memory needs without losing any data. You can use the **Time** view graph to examine peak memory use and determine if the system can provide enough memory for this scenario.

The **Time** view graph shows at what point the maximum memory usage was reached, how quickly it was reached, what allocations contributed to it, and how long memory usage was maintained at that level.

4.4 Advanced Topics

Detecting Leaks in the Boot Process

For VxWorks only, this example shows how to incorporate the Memory Analyzer target libraries into the kernel image to allow data to be collected very early in the boot process. This enables detection of leaks that might otherwise go unnoticed.

The following process is conceptually simple; putting it into practice depends on the complexity and flexibility of the build process employed by the user system:

1. Load the necessary application libraries in the bring-up process, or compile them directly into the kernel. The required libraries are the same as those loaded by Memory Analyzer when started from Workbench. These can be viewed by typing **moduleShow** into the Wind Shell after loading Memory Analyzer.
2. Insert a call to **Memory AnalyzerInit** with the proper parameters somewhere early on in the user bring-up code. This initializes the target-side code necessary for Memory Analyzer to operate, and allow the GUI to connect. To guarantee that enough time is available for the GUI to connect, it may be desirable to have the system wait for a few seconds at this point, if possible.
3. Load the GUI before booting the target system. If data is collected before the GUI is connected, and the target side queue overflows, data will be lost. But if the GUI is loaded first, it will connect as soon as the target side is initialized, and therefore no data will be lost.
4. Boot the system, then Memory Analyzer begins collecting data as soon as the target side is initialized.

Detecting Leaks in Processes

Both Linux and VxWorks support the classic Process Model. VxWorks calls its processes **Real-Time Processes (RTPs)**. All resources, including memory, in a process are recovered and returned to the operating system when the process is deleted.

Each process has its own memory region allocated to it when it is created. During process creation, some memory from the kernel memory heap may be added to the process memory region so that all tasks and modules may be loaded and executed.

When a process dies or is deleted, the memory that was allocated to the process from the kernel heap is recovered for use by other processes. However, it is not returned to the kernel heap and, therefore, appears as memory leaks in Memory Analyzer. In VxWorks, the **memShow()** routine also shows these apparent leaks.

These are not leaks, rather, they are a matter of book keeping and are completely harmless. Remember that the operating system recovers all process memory and other resources for reuse when a process dies or is deleted.

To find memory leaks within your processes, filter the collected data by enabling the **Show Outstanding Allocations Only** option (see [Aggregate Allocations Table](#), p.27), then look for your user tasks in the **Aggregate Allocations** table. In that table, concentrate only on allocations directly attributable to the routines in your task.

Additional Help

For additional advanced help in locating memory leaks, see the *Error Detection and Reporting* chapter (ED&R) in either the *Wind River Application Programmer's Guide*, or the *Wind River Kernel Programmer's Guide*. There are detailed references in this chapter to the many procedures and options for configuring, displaying, and clearing error records, and for configuring system response to fatal error records. Also described is a set of convenient macros you can use in your source code to generate error messages, and system response to fatal errors.

The Wind River compiler (Diab) contains yet another set of capabilities (RTEC).

5

Troubleshooting

5.1	Introduction	71
5.2	Messages	72
5.3	General Troubleshooting Tips	74

5.1 Introduction

This chapter addresses the following problem areas:

- Memory Analyzer status and error messages ([5.2 Messages](#), p.72).
- Typical problems ([5.3 General Troubleshooting Tips](#), p.74).

If you get error messages, or are having problems getting Memory Analyzer to work, check the error messages and troubleshooting tips in this chapter to see if they resolve your problems. If you are still unable to get Memory Analyzer to work, contact Wind River Technical Support.

5.2 Messages

Message traffic within Memory Analyzer, and with its external parts, is formatted and displayed in a variety of places. Status messages appear in a **Analysis Console** view (see [3.2.8 Analysis Console View](#), p.44). Warning messages can appear in the Console view, but more often appear in dialog boxes. Error messages are displayed in the **Console** view.

Target Errors

Some of the important status, warning, and error messages, including interpretation and helpful suggestions where needed, is included here.

- **Connection**

The following messages may appear at the left-hand end of the status bar:

Unconnected

Memory Analyzer is not connected to any target server.

Connecting to *targetServer*

Memory Analyzer is in the process of connecting to the *target server*, where *target server* is the name of your target server. If the message remains for more than a very short time, it probably means you need to reload the target libraries.

Connected to *targetServer*

Memory Analyzer has successfully attached to *target server*.

- **Overflow**

No errors

The message queue on the target that buffers allocation records has not overflowed.

Overflow

The message queue on the target that buffers allocation records has overflowed. Overflow counts are reported to the standard output of the analyzed task and to the Memory Analyzer Console. For more information on overflows, see [Detecting Leaks in the Boot Process](#), p.69.

- **Data Collection**

No errors

The message queue on the target that buffers allocation records has not overflowed.

WARNING: some data lost

Data has been lost, but collection is still proceeding. Select this message in the status bar to see a more complete description of why data was lost and recommended steps for avoiding data loss in the future.

ERROR: not collecting data

Data may have been lost, and an error has occurred that prevents any further data from being collected. Select this message in the status bar to see a more complete description of what went wrong and recommended solutions. Note that if you receive this message repeatedly, it may indicate a condition that requires contacting Wind River Technical Support.

Host Errors

In the process of starting and running Memory Analyzer, various error messages may appear at certain times. Some of these messages describe actual errors resulting from various issues. Some messages only describe potential errors, or errors in unrelated or non-crucial tasks. This section lists the major or most frequently encountered error messages in both the actual and benign categories.

VxWorks Errors

- **Loading**

If you have trouble loading the object files onto your VxWorks target, such as the error message:

API_FILE_NOT_FOUND

check the following:

- You are able to **ping** the target over the network.
- If you are using NFS, check that the file system is mounted (use **nfsDevShow**).
- Your target has permission to read the object files from the file server.

If none of these suggestions resolve the problem, it may be that your target system is slow, or has intermittent response. Try loading the Run-Time Analysis Tools modules manually using a shell window and the **ld** command.

- **Connection**

`Lost connection to target server.`

Memory Analyzer has lost its connection with the target. Possible causes include the following:

- The target was rebooted.
- The network connection was interrupted or disconnected.
- The target server is busy, or the machine running the target server is busy.

Linux Errors

- **insmod**

`Failed to execute insmod: No such file or directory`

This message in your target terminal window, indicates that the **insmod** and **rmmod** programs are not in your path, as noted in [GUI Issues on Linux](#), p.81.

5.3 General Troubleshooting Tips

This section organizes problem areas by the major components in which they occur.

Issues With the Target

- **Target Connection Lost**

If the target connection is lost, the message

`Lost connection to target server (mode)`

appears in the status bar, the status bar message changes to **Unconnected**, and the **File > Connect to** and **File > Reconnect** commands on the menu bar

become available. They remain available until you either successfully attach to a target or exit Memory Analyzer. The **Connect to** command lets you select any target, while **Reconnect** attempts to connect only to the previously connected target.

You may also receive the following message in the target shell:

```
Link ERROR: Broken Pipe
Error sending records, reconnecting...
```

If so, it means the target has replied back to the host, and the host has shut down the target. In this case, the error is possibly caused by the target not responding to the host within the specified target timeout period. You can adjust priorities and timeouts as follows, then retry.

Increase the priority of Memory Analyzer to a value just below the tWdbTask priority, and above the tNetTask priority. For instance, if tWdbTask priority is at 3 and tNetTask priority is at 50, set **Task Priority** in the **Connect to Target** dialog box to 9, reconnect, and try again.

You may also need to change the **Backend request timeout** value from the default 3 sec. to a higher number, such as 10. Do this (with your target disconnected) by right-clicking your target server in the **Remote Systems** view, then selecting **Properties** to open the **Target Connection** dialog box. In this dialog box, use the **Advanced target server actions** group in the **Target Server Actions** tab view to modify the timeout value as indicated above.

▪ **Call Stack Display (VxWorks Only)**

Memory Analyzer does not appear to be displaying the proper call stacks for the memory-allocation records.

Cause #1 — The target server was not started with the **-A** option.

Solution #1 — You must start the target server with the **-A** option. This ensures that the target server loads local symbols in addition to global symbols. If this option is not selected, the call stack traces pick the nearest global symbol for calls from local symbols. For more information, see [2.2 Requirements](#), p.10.

Cause #2 — You did not manually load libraries with local symbols, so Memory Analyzer instead shows function names that are the nearest global symbols.

Solution #2 — Make sure you load your libraries with local symbols using the

```
ld 1 < ...
```

command.

Cause #3 (for x86 targets only) — Your target kernel was possibly compiled with frame pointers disabled.

Solution #3 — Frame pointers enabled is the default with the compiler, but make sure you did not disable them when you last compiled. If you did, you must recompile your code with frame pointers enabled (see [VxWorks](#), p.10).

- **Target Kernel Start and End Addresses**

When starting Memory Analyzer on VxWorks, and DFW is unable to determine the target kernel text start or end address, a dialog box opens with the following warning:

```
Unable to locate the start and/or the end of kernel text address,  
which the tool needs in order to successfully connect to the target.  
Please enter the values manually below:
```

Enter the start and end addresses manually in the fields provided, and continue the startup process. However, if you do not know the exact layout of your target memory and cannot supply correct values, you must cancel the connection and rebuild your VIP project, adding to it the following symbols:

```
wrs_kernel_start_text  
wrs_kernel_end_text
```

This enables DFW to provide the needed addresses.

- **Degraded Performance while Running RTPs**

If your target code contains RTPs, and you start them running only after you have connected Memory Analyzer, you may experience an unacceptable level of slow response from Workbench and the target. This could be manifested by the RTPs taking a very long time (up to several minutes) to become fully operational, and even longer for symbols to begin showing up in Memory Analyzer.

You may also notice that some symbols are unresolved when the target code is first started. This is because the first calls into the new RTP's memory library are captured by the Memory Analyzer GUI before the RTP task ID and symbols have been registered by Workbench.

Under certain conditions you may experience an even greater lack of response. If your RTP spawn time limit is short (say 30 seconds or less), you will see the message,

```
Failed to launch RTP name.
```

If the spawn time limit is longer and the RTP actually launches, you may see,

```
Target OS object not found.
```

A simple workaround is to increase the priority of the RTP and try again. However, you *must* follow these steps to be successful:

- a. Disconnect your target if it is connected.
- b. Edit the following values in the **Advanced target server options** group of the **Target Server Options** tab view in the target **Properties** dialog box:
 - Set the RTP spawn time limit to **120** seconds or greater, and the backend request time limit to **30** seconds.
 - Increase the RTP's initial task priority from the default 100 to a value of about **60** (higher than MemLink but lower than the network task) to enable the RTP to execute cooperatively with Memory Analyzer.

These are recommended values. Note that the time outs may have to be increased and/or the RTP priority decreased if the recommended values do not work.

This procedure works because the slow-loading RTPs are loaded, or nearly so, before Memory Analyzer starts and begins its memory-intensive communication activities over the target connection. This workaround also prevents the unresolved symbols behavior described above.



CAUTION: If you do not attend to these items, the RTP initialization task may not receive sufficient CPU time to complete its execution before the RTP spawn time limit expires, and thus causing the host to stop all tasks running in the RTP.

For more information on this topic, see *Wind River Workbench User's Guide: RTPs and Shared Libraries from Host to Target*, and also check Workbench online help for **spawn time limit** while building your RTP task.

▪ **Unknown Signals**

Memory Analyzer uses real-time signals within the context of analyzed processes. This can have the following repercussions:

- If you are running a process under a debugger, and it reports catching unknown real-time signals, you can just ignore these messages and continue.
- The signals used by Memory Analyzer are hardcoded as **SIGRTMIN+3** through **SIGRTMIN+5**. If your target application installs handlers for these signals, the results will be unpredictable. Note that the GNU libc already uses **SIGRTMIN** through **SIGRTMIN+2**, so in general you are better off referencing your signal numbers from **SIGRTMAX**.

- **insmod and rmmod**

The programs **insmod** and **rmmod** must be in your user PATH because **MemAgent** depends on them. They must also have execute permission (logging on as **root** gives this permission by default).

- **Verbosity Effects**

Many status messages are hidden during normal operation. If you are having problems, increase the verbosity of the host and/or target side components to gain some insights. Verbosity can be set when you launch Memory Analyzer, using the **MemAgent Setup Options** dialog box, displayed when you connect to your target.

- **MemAgent trouble loading kernel modules**

If **MemAgent** has trouble loading the necessary kernel modules, examine your syslog for lines beginning with **Run-Time Analysis Tools**. These messages will provide more detail about the failure.

- **Processes and Shared Memory Reporting**

Top reports the memory footprint of instrumented processes increasing over time. The MemAgent process uses shared memory to communicate with analyzed processes. As the analyzed processes place data into the shared memory region, they must periodically map more memory into their address space. So although the address space reports using more memory, actual memory usage does not increase.

- **kgdb Support**

The Memory Analyzer target binaries can handle non-Wind River Linux kernels with no user intervention, whether they support **kgdb** or not.

Wind River Linux kernels support **kgdb** by default. The Run-Time Analysis Tools target binaries can handle those kernels with no **kgdb**-related user intervention. However, you may need to rebuild KAL For details, see [A. Kernel Abstraction Layer \(KAL\)](#).

If you have any doubts about your particular Wind River Linux kernel, boot the kernel and check the **dmesg** or the **/var/log/messages** output for any mention of **kgdb**. If there is any mention of it, your kernel *does* support it.

For Wind River Linux kernels that do not support **kgdb**, you must execute the following steps in order to use Memory Analyzer with your Wind River Linux target kernels:

- a. Rename **Memory AnalyzerModule.ko** to **Memory AnalyzerModule.ko_orig**
- b. Copy **Memory AnalyzerModule.ko_no_kgdb** to **Memory AnalyzerModule.ko**

To support Wind River Linux and **kgdb**, module versioning has been disabled in the build of target kernel binaries. Because of this, you may see the following system messages during target reboot, but they may be disregarded:

```
KAL: no version for "struct_module" found: kernel tainted
```

The above message is displayed because the default KAL is compiled with no version info. If you recompile KAL, you do not see this. If you do not have versioning turned on in your running kernel, you also do not get this message.

```
KAL: module license 'ScopeTools License Agreement' taints kernel
```

You will always see this message.

```
KAL: No versions for exported symbols. Tainting kernel.
```

The above message is only displayed if KAL is not recompiled, or you have versioning turned on in your running kernel.

```
MemoryAnalyzerModule: no version magic, tainting kernel.
```

This message is generated for all Wind River Linux 2.6 kernels.

If you have recently modified the Run-Time Analysis Tools target binaries you are using, be sure to reboot your target to start with a clean running target kernel system before using Run-Time Analysis Tools. If you have problems with Run-Time Analysis Tools target binaries for one of the tools, like Memory Analyzer, and you want to switch to using a different tool, for example ProfileScope, you should reboot your target to clean out anything left in the target memory by the first Run-Time Analysis Tools tool.

▪ **Cannot Select a Linux Process**

If you try to select a process in the **Process Selection** dialog box, and get a **Failed to patch** notice due to routines such as **kmalloc** and **vmalloc**, the reason for the error is that the process is a kernel thread. For Linux targets, Memory Analyzer supports the analysis of user-space processes only. Dynamic allocations that occur in a kernel memory region cannot be analyzed by Memory Analyzer.

Issues With the GUI

Viewing Source Code

```
Module name not available. Unable to show source code.  
function name is not found in directory.  
No debugging information for object module module.
```

You may see one of these errors when trying to display source code (as discussed in [3.2.7 Source Code Viewer](#), p.41). If the module name cannot be found, there is no fix available, and source code cannot be displayed. For the debugging information error, make sure your source code has been compiled with debugging enabled.

Generally, **View Source Code** functionality needs some configuration and setup before it can work properly. Check to be sure that the following steps have been taken:

- a. Modules must be compiled with debugging information. If they are not, a message will inform you that the module does not contain debugging information.
- b. The source paths to the modules must be properly configured in the **Preferences** dialog box; for detailed instructions, see [3.2.10 Preferences Dialog Box](#), p.46. This is only necessary if the path to the source code as mounted on the host machine is different from the path where the module was compiled. For example, this can occur when the module is compiled on a Solaris machine but the host is a Windows machine.

View Source Code functionality only works with object files containing Wind River accepted symbol and debugging information. Some compilers and target architectures may not generate this information.

Memory Analyzer Database Problems

▪ Database Deleted

When you start Memory Analyzer after having exited at least once previously, you will be prompted in the login procedure to save the data generated by the previous Memory Analyzer session. If you try to access the Memory Analyzer database and it cannot be found anywhere on disk, you may have neglected to save the database before proceeding to start up Memory Analyzer. Each Memory Analyzer invocation prompts you to save the previous session's database - it is not automatically saved, and will be lost if you do not save it.

For more information on the Memory Analyzer database, see [Database Tab View](#), p.51.

- **Memory Analyzer Will Not Connect**

If Memory Analyzer will not connect, it may be that the previous database file cannot be deleted for some reason. If the previous database file is in use (for instance, if you are running SQLITE on the database), then Memory Analyzer will not connect.

GUI Issues on Linux

5

- **Symbol Resolution**

If the data collected by Memory Analyzer includes incorrect symbols, there are a few common possibilities:

- The library or executable containing the incorrect symbols has been either partially or completely stripped. Memory Analyzer parses the ELF symbol table to correlate process addresses with actual functions. If the symbol table no longer contains the function that corresponds to a set of addresses, those addresses will be attributed (incorrectly) to the next nearest symbol.
- Memory Analyzer is parsing the wrong file for symbols. The tool searches each of the object paths given by the user, in order, to find the file which has the symbol information needed to resolve the raw addresses. If a library or application has the same name as the application you are analyzing, and resides in an earlier entry in the object path list, it will be parsed instead of the correct one. Depending on how similar the two files are, this can result in incorrect offsets or incorrect functions altogether.

General Tips

This section lists suggestions for dealing with the following problems:

- Connecting to a target.
- Staying connected to a target.
- Getting data from a target.
- Unexpectedly dropping a connection to a target.
- Run-Time Analysis Tools target modules and/or user-space programs and daemons crashing.
- A target crashing.

Suggestions

The following general guidelines are offered to help keep you from problems, and as procedures to follow if you do have problems:

- If you have recently modified the Run-Time Analysis Tools target binaries you are using, be sure to reboot your target to start with a clean running target kernel system before using any of the Run-Time Analysis Tools. If you have problems with Run-Time Analysis Tools target binaries for one of the tools, like Memory Analyzer, and you want to switch to using a different tool, for example ProfileScope, you should reboot your target to clean out anything left in the target memory by the first Run-Time Analysis Tools tool.
- When you enter your parameters into the **Connect To Target** dialog box be sure that you set the **Verbosity** to 3, which is the most verbose. This causes trace messages and other diagnostic info to be displayed in the target terminal window where you started it.
- Other important Run-Time Analysis Tools-related system-level diagnostic messages might appear in the following places:
 - The `/var/log/messages` file.
 - The `/var/log/kern.log` file.
 - The `dmesg` program.
 - The `syslog`.

Be sure to collect these messages with the **Help > Collect Log Files** option, and submit them to Wind River Technical Support with your Run-Time Analysis Tools-related target system issues.

- In the Memory Analyzer perspective there is an **Analysis Console** window. You can increase the verbosity in this window by selecting the drop-down arrow to the right of the yellow "!" icon in the toolbar and selecting **Debug-hi**.

Known Issues and Workarounds

The following are issues encountered in setting up and testing various target and host combinations, along with suggested workarounds, if any

- **View Source feature displays incorrect offset**

The **View Source** feature may not display the correct offset into the currently running or highlighted function.

- **View Source feature stops working**

The **View Source** feature of Memory Analyzer may stop working after a while and new symbols are no longer resolved. This can occur if you choose to view source on an object that has both a standard and dynamic symbol table, (that is, the `symtab` and `dynsym` sections) then request symbols from an object that has only one of the two sections; typically this happens if you view source using Memory Analyzer relatively soon after connecting it to a target. To work around this issue, try the following:

- Use only objects that either include both symbol tables, or only use the same symbol table type.
- Only use the View Source feature after the tool has analyzed the bulk of the target objects you expect to be used.
- Use View Source on a function that resides in an object that only has a dynamic symbol table directly after Viewing Source on a function residing in an object that has both symbol tables.
- Disconnect and reconnect the faulting tool when the error occurs.

- **WR Linux threads are all analyzed**

For Linux 2.4 target kernels, threads are treated as regular processes. So if you run a program that has four threads in it, there will be four entries for that program in the Memory Analyzer **Process Selection** dialog box. Each entry has the program name, and also has a different process ID (pid). If you select any of these threads to be analyzed with Memory Analyzer, all of the threads will be analyzed.

For Linux 2.6 target kernels, threads are treated as lightweight processes that are grouped together under the pid of the initial process. So in the **Process Selection** dialog box there is only one entry for a multi-threaded process. Selecting that one process entry to be analyzed causes all threads running for that process to be analyzed.

- **Memory Analyzer and debugger running together may crash process**

Using Memory Analyzer on a process that is currently being executed by the debugger can cause the process to crash. The workaround is to use either the debugger or Memory Analyzer, but not both together. This happens on both PowerPC and x86 targets.

- **memrun fails with stripped programs**

memrun fails to collect data for stripped programs. A stripped program does not have a **main()** symbol, and thus **memrun** cannot hold the program at main

for patching before the program can run and collect memory allocation data for it.

- **VxWorks simulator configuration for TCP/IP**

VxWorks simulators require special configuration to enable a TCP/IP connection. If you try to connect Run-Time Analysis Tools to a simulator using TCP/IP, you might see one of the following messages:

```
Target server could not get ip address of target.  
SVR_TARGET_UNKNOWN  
  
Failed to get target's IP  
LINK ERROR: Failed to connect to target, giving up  
  
Unable to obtain IP address from the target server  
vxsim0@xxxx.
```

All of the tools offer the **wtx** connection type also. You can use that instead of TCP/IP to connect to your simulator. If you want to use Run-Time Analysis Tools with TCP/IP to connect to a simulator, you may need to rebuild and reconfigure the simulator to handle TCP/IP. For more information, refer to the *Wind River Simulators User's Guide*.

- **Only kernel level heap recorded by VxWorks simulator**

On a VxWorks simulator, Memory Analyzer records only invocations of the kernel level heap allocation functions.

- **High-memory PPC target may crash after some errors**

If you are using a high-memory ppc VxWorks target, and if you see errors on your target system console such as **Offset for Vector 740 out of range**, the **wdbagent** may crash shortly thereafter. The explanation is that by default, PowerPC target kernels support 26-bit vector addressing. High-memory PowerPC targets have larger memory areas and can use 32-bit extended addressing. Kernels must be compiled with the **INCLUDE_EXC_EXTENDED_ADDRESSING** option for these targets to allow this extended addressing to happen. Memory Analyzer libraries are normally loaded into high memory and thus require 32-bit extended addressing support. Therefore, if you are using a high-memory PowerPC target, you must compile your kernel with the option mentioned above. For details, refer to [VxWorks](#), p.10.

- **Memory Analyzer does not work on Linux with wrDiagnostics**

Memory Analyzer does not work on a Linux platform with **wrDiagnostics**. It displays the following errors on the target during Memory Analyzer launch:

```
*ScopeTools ERROR * init_module(396) * failed to install trap-handler  
KAL: file operation weren't properly unregistered!  
KAL: unregistering lingering ScopeTools drivers...
```


A

Kernel Abstraction Layer (KAL)

[A.1 Introduction 87](#)

[A.2 Basis for Need 87](#)

[A.3 Procedure 88](#)

[A.4 Known Issues and Workarounds 92](#)

A.1 Introduction

The kernel abstraction layer, or KAL, is a script used by a **Linux** target installation to resolve any binary incompatibilities that might exist between the Memory Analyzer kernel and a modified Linux kernel.

A.2 Basis for Need

If you know any of the following conditions exist, you *must* relink the Run-Time Analysis Tools KAL.ko kernel module with your Linux kernel:

- You are using Run-Time Analysis Tools for the first time on a Linux target.

- You have modified or reconfigured your Linux target kernel in any way.
- You are using a Linux distribution from another vendor on your target.

For any of the above conditions, you must recreate the Run-Time Analysis Tools KAL.ko kernel module to match your Linux kernel, and to resolve any binary incompatibilities. You do this using a special KAL makefile and kernel module supplied with Workbench. This module performs the following actions.

- Copies the correct Analysis Tools target agents to your rootfs staging area.
- Builds a KAL.ko module that matches your current kernel.

The easiest way to tell if you need to make changes to your KAL configuration is to build KAL using it. If you get compilation errors this usually indicates that the KAL configuration needs to be modified. Looking at the source for KAL.c will usually make it clear which feature specifications need to be changed.

A.3 Procedure

If you determine that you must relink the Run-Time Analysis Tools KAL.ko kernel module to your Linux kernel, follow the steps outlined in the following sections.

Setup

The makefile and Linux kernel are both obtained and used by the script file:

```
installDir/scopetools-6.0/target/src/kal/buildKAL.sh
```

This script compiles the shared Run-Time Analysis Tools kernel module (KAL.ko) to work with your customized Linux kernel using your gnu compiler or cross-compiler. You need to provide information to this script in order for it to compile your new KAL.ko correctly. This information is described when you run the script.

You can run the buildKAL.sh script on any host system that has all of the following criteria:

- Linux- or Unix-like environment
- bash shell
- access to the proper toolchains and kernel source/config



NOTE: The buildKAL.sh script uses the bash shell interpreter program and expects it to be in a file system location as on a Linux system (/bin/bash). This location may be different on a Solaris system.

If you try to run the buildKAL.sh script on a Solaris system and you get an error message about **buildKAL.sh: not found**, then you will need to invoke the bash shell interpreter along with the buildKAL.sh script as follows:

```
bash buildKAL.sh
```

This should successfully start up the buildKAL.sh script.

There are also KAL configuration files available in the **configs** sub-directory for several predefined configurations. You can use one of these, or you may need to create your own KAL configuration. To create your own, choose a similar configuration file and copy it, then modify it as needed for your configuration. For example, if you wanted to create a KAL configuration for a Linux-2.6.11 kernel, copy the predefined config-2.6.10 to config-2.6.11 and edit it. Modify the kernel version number and any other feature specifications as needed. In many cases, no changes other than the version number would be needed.

Wind River Linux 2.0 Targets

The process of initially building the KAL.ko kernel module is automated when you build your target work space environment.

For a **Workbench Interface**, the commands are as follows:

- **Create** a new platform project
- **Build All**
To do this, in the **Project Explorer** view, expand your platform project, right-click **all**, then click **Build Target**. This will build KAL and include Run-Time Analysis Tools on the root file system.

or

- **Build Run-Time Analysis Tools**
To do this, in the **Project Explorer** view, expand your platform project, open **User Space Configure**, select the Run-Time Analysis Tools package, select the **Targets** tab, then click **Build**.
- **Build fs**
Back in the **Project Explorer** view, right-click **fs**, then select **Build Target**.

For a **Command Line Interface**, the analogous commands are as follows:

- **configure**
- **make all**

or

- **configure**
- **make linux** (optional)
- **make -C build scopetools** (optional)
- **make fs**



NOTE: For a **small** root file system (see [Wind River Linux 2.0 Targets](#), p.5), the **configure** command in either instruction above becomes:

configure --with-template=extra/scopetools...



NOTE: The **make -C build scopetools** command in this procedure is optional, and will be done for you automatically by the **make fs** command if it does not exist.

These commands are used to create the target workspace on your host, then create the root file system, and build all the kernel modules (**Build All**), or create the root file system and use the default kernel (**Build fs**, a shorter process). This work is all done in a staging area on your host system without copying anything to the target.



NOTE: This automation of the KAL.ko kernel generation and root file system copying is available only for **Wind River Linux 2.0** targets.

If you have modified your kernel and want to rebuild it, be aware that rebuilding your kernel does not automatically rebuild the Run-Time Analysis Tools kernel, nor does it copy the Run-Time Analysis Tools binaries back to your target. After you rebuild your modified kernel, execute the command:

make scopetools.rebuild

then manually copy the **scopetools-6.0** directory from the *buildDir/build/* directory to your target root file system.

Other Linux Targets

The procedure is as follows:

1. After determining that your present KAL configuration needs to be modified following the suggested scenario outlined in [A.2 Basis for Need](#), p.87 above, make the necessary edits.
2. Run the script using the following command:

```
buildKAL.sh
```

The script asks you questions concerning the availability of your gcc compiler (or sometimes a cross-compiler) for your target, and the existence of a Linux kernel configured and built for your target. This step compiles and links the kernel module (KAL.ko) to work with your customized Linux kernel using your gnu compiler or cross-compiler, and places the shared Run-Time Analysis Tools kernel module in the current directory. Note that entering a question mark ("?",) at most prompts causes help for that prompt to be displayed.

You can run the buildKAL.sh script on any host system that has all of the following criteria:

- A Linux or Unix-like environment.
- A bash shell.
- Access to the proper toolchains and kernel source/config.



NOTE: The buildKAL.sh script uses the bash shell interpreter program and expects it to be in a file system location as on a Linux system (/bin/bash). This location may be different on a Solaris system.

If you try to run the buildKAL.sh script on a Solaris system and you get an error message about **buildKAL.sh: not found**, then you must invoke the bash shell interpreter along with the buildKAL.sh script as follows:

```
bash buildKAL.sh
```

This should then successfully start up the buildKAL.sh script.

There are also KAL configuration files available in the configs sub-directory for several predefined configurations. You can use one of these, or you may need to create your own KAL configuration. To create your own, choose a similar configuration file and copy it, then modify it as needed for your configuration. For example, if you wanted to create a KAL configuration for a Linux-2.6.11 kernel, copy the predefined config-2.6.10 to config-2.6.11 and edit it. Modify the kernel version number and any other feature specifications as needed. In many cases, no changes other than the version number would be needed.

3. Copy the new KAL.ko target-binary to your target file system in to same location where you copied the other Run-Time Analysis Tools binaries for your target.
4. Be sure that when you change which target binaries you are using that you reboot your target to start with a clean running target kernel system.

KAL is designed to be adaptable to a wide range of Linux kernels. Unfortunately, it is not possible to adapt to all of the differences between various Linux kernels or to anticipate all of the possible future modifications. Please let us know if you encounter a kernel configuration for which you are unable to create a working KAL configuration.

A.4 Known Issues and Workarounds

The following are some things to be aware of when building and using the KAL.ko file:

- **Target binaries and target kernel do not match**

If you see the error output when connecting to your target using one of the Run-Time Analysis Tools

```
insmod: error inserting 'KAL.ko': -1 Invalid module format
```

or the Oops in the following output

```
Found data_access@c000140c
executing: insmod
:insmod
:KAL.ko
KAL: no version for "struct_module" found: kernel tainted.
KAL: module license 'ScopeTools License Agreement' taints kernel.
KAL: No versions for exported symbols. Tainting kernel.
Oops: kernel access of bad area, sig: 11 [#1]
PREEMPT
NIP: C004420 LR: C003DAF0 SP: DF299F10 REGS: df299e60 TRAP: 0300
Tainted: PF
MSR: 00029000 EE: 1 PR: 0 FP: 0 ME: 1 IR/DR: 00
DAR: E106C000, DSISR: 00000000
TASK = ded1c790[1158] 'insmod' THREAD: df298000
Last syscall: 128
GPR00: 00000000 DF299F10 DED1C790 E106C000 00000553 0000001F E1062280
00000004
```

```

GPR08: DFAB596C 000000A0 00000000 C085D810 22000082 10096B1C 1001CED8
00000000
GPR16: 7FFFF580 7FFFF910 00000003 FFFFFFFE 000000C8 FFFF9008 00000000
0FFC9454
GPR24: 7FFFE436 10091F38 00000002 C02F0000 30000008 10091F38 C02F1FA0
E106B7E0
NIP [c0004420] flush_icache_range+0x24/0x50
LR [c003daf0] sys_init_module+0xa4/0x42c
Call trace:
[c0001d88] ret_from_syscall+0x0/0x70

```

it means the Run-Time Analysis Tools target binaries you are using on your target do not match the kernel you are running on your target, and thus you must run the buildKAL.sh script in order to generate new Run-Time Analysis Tools target binaries.

- **Notification that KAL successfully loaded**

If you see the KAL-related messages during connection to your target

```
KAL is already loaded
```

or

```
Module KAL loaded, with warnings
```

or (if verbosity = 3 is enabled)

```

executing: insmod
:insmod
:KAL.ko

```

and nothing else related to KAL after this, then KAL has been loaded successfully.

Note that this only means that KAL.ko has successfully loaded. If there is a binary compatibility difference between your KAL.ko and the kernel running on your target, you may yet encounter runtime problems or crashes. So be sure to run the KAL script if you make any changes to your kernel.

- **Error when running KAL.sh script**

When you run the buildKAL.sh script, you might get the following cc1 error:

```

make: Entering directory
`/springboard/RC5_CDs/wrlinux-1.1/build-
wrs_powerquicci_82xx/dist/linux-2.6.10'
CC [M] /kal/kal/KAL.o
<unknown> tried to exec cc1 but failed (No such file or directory)

```

There are two ways to specify a directory path to your GNU target-specific compiler programs, as in these examples:

```
/toolchains/PNE-1.1-FCS/bin  
/toolchains/PNE-1.1-FCS/i586-wrs-linux-gnu/bin
```

Always use the first example. The GNU compiler programs in the first example know where each other are located. In the second example they do not, and you will get the **cc1** error above. Note that with the first instance you definitely must specify a target-arch-specific prefix for your GNU compiler programs.

- **When connecting to your target, some messages can be ignored**

When you try to make a connection from the GUI to your target, KAL and your Run-Time Analysis Tools target modules are loaded into the kernel. Note that the following tainting messages are not errors and can be ignored:

```
Jul  6 17:37:36 oahu kernel: KAL: no version magic, tainting kernel.  
Jul  6 17:37:36 oahu kernel: KAL: module license 'ScopeTools License  
Agreement' taints kernel.
```

or

```
Warning: loading KAL.ko will taint the kernel: non-GPL license -  
ScopeTools License Agreement  
See http://www.tux.org/lkml/#export-tainted for information about  
tainted modules  
Warning: loading KAL.ko will taint the kernel: forced load
```

When you get module-loading errors, you might also get system messages in the system locations listed in [5.3 General Troubleshooting Tips](#), p.74.

In the case of KAL not loading successfully, be sure to send the **.config** file for your target kernel to Wind River support personnel.

B

Event Dictionary

System Viewer Events

These System Viewer events posted by Memory Analyzer include the function name, address, and sequence ID number of each corresponding trace record.



memStart

- **Possible Causes**

System or application code called **MemScopeInit()** manually, or if you started Memory Analyzer from the Workbench toolbar, it is called automatically.

- **Task State Effects**

None.

- **Information Collected**

Event Parameter	Sample Data	Description
eventName	memStart (39200)	The name of the event associated with this icon.
bufferSize	2,000	Number of messages queued on the target before being sent to the GUI.
stackDepth	8	Depth of the call stack that will be collected for every trace point.

useWTX	1 or 0 (boolean)	Using WTX to transmit data from target to host? (1 = yes, 0 = no)
--------	------------------	---



memStop

- **Possible Causes**
System or application code shut down the Memory Analyzer GUI on the host, or the **MemScopeTargetExit()** function was executed.
- **Task State Effects**
None.
- **Information Collected**

Event Parameter	Sample Data	Description
eventName	memStop (39201)	The name of the event associated with this icon.



memAlloc

- **Possible Causes**
System or application code executed the **malloc()**, **calloc()**, **valloc()**, **memAlign()**, **memPartAlloc()**, or **memPartAlignedAlloc()** function, or the allocation function in a Custom Malloc/Free scheme, if user implemented.
- **Task State Effects**
None.
- **Information Collected**

Event Parameter	Sample Data	Description
eventName	memStart (39202)	The name of the event associated with this icon.
address	adress=0x1ffa428	The memory address of the beginning of the patched function.

partition	partition=0xd3674	The identifier of the partition in which the memory was allocated.
size	size=0x50 (80)	Number of bytes of the usable memory allocated, excluding VxWorks block header, plus any required alignment padding.



memFree

- **Possible Causes**
System or application code called the **free()**, **cfree()**, or **memPartFree()** routine, or the free function in a Custom Malloc/Free scheme, if user implemented.
- **Task State Effects**
None.
- **Information Collected**

Event Parameter	Sample Data	Description
eventName	memStart (39203)	The name of the event associated with this icon.
address	adress=0x1ffa480	The memory address of the beginning of the patched function.
partition	partition=0xd3674	The identifier of the partition in which the memory was allocated.



memFailedAlloc

- **Possible Causes**
System or application code executed the **malloc()**, **calloc()**, **valloc()**, **memAlign()**, **memPartAlloc()**, or **memPartAlignedAlloc()** function, or the allocation function in a Custom Malloc/Free scheme, if user implemented.
- **Task State Effects**
None.

Information Collected

Event Parameter	Sample Data	Description
eventName	memFailedAlloc (39204)	The name of the event associated with this icon.
partition	partition=0xd3674	The identifier of the partition in which the memory allocation was attempted.
size	size=0x75b4abf	Number of bytes requested.



memOverflow

Possible Causes

Data is being generated on the target faster than it can be transmitted to the host. You may need to restart Memory Analyzer with a larger buffer to reduce the number of records being generated.

Warning: The most recently generated data is being lost.

Task State Effects

None.

Information Collected

Event Parameter	Sample Data	Description
eventName	memOverflow (39205)	The name of the event associated with this icon.

C

Glossary

This glossary contains definitions for some of the common terms used throughout this manual.

allocation record (AR)

The full call stack for every memory allocation event recorded by the collection agent and forwarded to the host computer.

call stack

The list of nested subroutine calls that lead up to a memory-allocation call. For each memory-allocation record, Memory Analyzer includes the call stack for that call.

collection agent

The part of Memory Analyzer that runs on the target.

graphical user interface (GUI)

The collection of computer programs and the media-oriented screens, windows, dialog boxes, menus, and icons they produce that provide for enhanced human-computer interactions with no, or minimal, keyboard input.

heap

The part of memory reserved and used for data variables generated and used by the computer program (in contrast to **Program** and **Stack** memory).

host

The computer on which the Memory Analyzer GUI is running, which receives and processes the allocation record data collected from the agent machine.

inferior process (Linux only)

Any process that has been patched by Memory Analyzer to have its data collected and analyzed.

kernel abstraction layer (KAL) (Linux only)

A layer of source code containing all dependencies on kernel constructs, that is compiled and linked to a proprietary object kernel, and through which kernel resources are accessed.

memory (heap) corruption

Heap corruption occurs when a program writes into a place in memory that does not belong to it.

memory leak

The condition where one or more memory blocks allocated by a program are not freed, with the tendency to build up to a large amount of allocated but unused, and therefore inaccessible, memory.

module

A collection of functions, such as a library or executable.

offset

In a call stack, this is the memory distance (in bytes) from the start of the current routine to where the call to the next function occurs.

partition

For VxWorks only, a section of memory configured and set aside by a running program.

patch

A process for changing run-time code dynamically, without compilation.

perspective

Any of the view windows within the Workbench Eclipse-based IDE.

polling rate

When there is a lull in data to be sent from the target to the host, this is the length of time the data-transfer process on the target waits before checking the buffer again for data to be sent.

process

See **task**

routine

A self-contained code module that can accept input, execute, and produce output; used interchangeably with **function**.

sticky button

A toolbar icon that remains visually depressed when you select it, sustaining the action until you select it again, turning off the action.

task

The program module currently being executed. See also **process**.

verbosity

Controls the type and number of messages generated by the target or GUI and displayed either in the target console or the GUI console.

view source code

Displaying the source code for the selected memory allocation/deallocation. For detailed information, see Sections [3.2.7 Source Code Viewer](#), p.41.

Index

A

- A option
 - needed for symbol loading 10, 75
 - troubleshooting 75
 - Warning 10
- agent
 - collection agent architectural description 3
- Aggregate Allocations table
 - call stack 28
 - customizing 29
 - description 27
 - task 28
- Aggregate view
 - Aggregate Allocations table 27
 - description 26
 - elements 26
 - Individual Allocations table 30
- allocation record (AR)
 - defined 99
 - technical note on patching 2
- AMP environment, WDB_TIPC 12
- architectural summary 3
- automatic
 - launching 13
 - System Viewer support 60

B

- bash shell interpreter program 89
- building
 - detecting leaks in boot process 69
 - ensuring WDB support for VIP 12
 - for PowerPC target 10
 - for real-time processes (RTPs) 10
 - KAL.ko kernel, issues with 92
 - KAL.ko kernel, procedure for 88
 - Linux+kgdb support 78
 - root file system for Linux 2.0 5
 - root file system, binary files 13
 - target kernel start/end address 76
 - x86, with frame pointers off 10

C

- call stacks
 - defined 99
 - depth 29
 - in Aggregate Allocations table 28
 - in Time view 35, 39
 - offset 100
- collection agent
 - architectural description 3
 - defined 99
- Color Palette option

- in Preferences dialog box 48
- colors, modifying 48
- common view elements
 - Memory Analyzer menu item 56
 - menu bar 56
 - Open Perspective menu item 57
 - Preferences menu item 59
 - Show View menu item 58
 - Window menu item 57
- communication link, target-to-host 4, 5
- connection status message 72
- Console
 - verbosity, compared to target 15
- Console view description 44
- correlation, among main views 31, 33, 36, 40
- creating a snapshot 53
- customizing the Aggregate Allocations table 29

D

- database files, viewing 5, 55
- debugging, viewing source code 80
- demo program, starting with MemScopeDemo 18
- Details table
 - call stack 35, 39
 - description 35
 - partition 35
 - timestamp 35, 39
- Details Viewport view
 - description 40
 - Show Module Names option 41, 44
 - Show Offsets option 41, 44
- detecting
 - memory hogs 68
 - memory leaks 64
- DFW 19
 - in Linux architecture 4
 - in VxWorks architecture 4
- dfwserver 19
- dialog boxes
 - Connect To Target - VxWorks 29
 - Include Wizard (shared data region) 11
 - Preferences 46
 - Process Selection 27, 83

- Properties (target server) 10

E

- error
 - dynamic allocations cannot be analyzed 22
 - messages 73
 - target server restarted 74

F

- features list 6
- Fragmentation view
 - description 36
 - fragmentation map, description 37
 - map blocks, resolution 37, 39
 - map colors 38
- frame pointers
 - call stack display debugging 76
 - must be enabled for x86 targets 10
- functions
 - see also routines
 - memPartAlignedAllocInternal() 4
 - memPartFreeInternal() 4

G

- graphical user interface (GUI)
 - defined 99
 - Memory Analyzer 23
- graphs, in Time view 34

H

- heap corruption, defined 100
- heap, defined 99
- host, defined 100
- host-side modules 4, 5

I

INCLUDE_SHARED_DATA 11
 insmod 13, 78
 installation, testing 16
 Interface Options, view 46

K

Kernel Abstraction Layer (KAL)
 see also Appendix A
 determine need to relink 87, 88
 how to tell if you need to update it 88
 procedure for building 89
 requirement for a Linux target 13
 requirement for Linux target 13
 kernel, cannot analyze memory allocations 22
 kgdb
 checking for support 78
 what to do if not supported 78

L

launching Memory Analyzer, automatically 13
 Linux
 2.4 kernel, thread analysis 22
 2.6 kernel, thread analysis 22
 targets, user-space only supported 22, 79
 loading
 configuration file 47
 snapshot file 54

M

Main Window
 see also Aggregate view
 elements 26, 32
 map blocks, resolution 37, 39
 Memory Analyzer
 architecture 3

 features 6
 list of views 24
 overview 2
 memory hogs
 detecting 68
 problem solving scenarios 68
 memory leaks
 analyzing the call stack 66
 defined 100
 detecting 64
 in the boot process 69
 problem solving scenario 63
 memory peak usage
 problem solving scenarios 68
 memPartAlignedAllocInternal() 4
 memPartFreeInternal() 4
 MemScopeDemo function 18
 menu bar 56
 menu items
 Memory Analyzer 56
 Open Perspective 57
 Preferences 59
 Show View 58
 Window 57
 messages
 error 73
 status 73
 modifying colors 48
 modules
 defined 100
 host-side 4, 5
 names, show 41, 44
 Show Modules option 47
 target-side 4, 5

N

NFS directory structure, caution 5

O

offsets

- defined 100
- show 41, 44
- options
 - A, needed for symbol loading 10
 - A, troubleshooting 75
 - Color Palette 48
 - extended vector addressing, for PowerPC 10
 - Show Modules 47
 - Show Offsets 47
- overflow status message 73
- overview 2

P

- partitions
 - defined 100
 - in Time view 35
- patching
 - defined 100
 - described 21
 - Warning, about MemAgent 21
- PATH, variables that must be in 13, 78
- polling rate, defined 101
- Preferences dialog box
 - description 46
 - saving to a file 47
 - tab views 46
- Preferences dialog box, options
 - Color Palette 48
 - Show Modules 47
 - Show Offsets 47
- Preferences dialog box, tab views
 - Aggregate View 48
 - Database 51
 - Fragmentation 50
 - General 47
 - Time 49
 - Tree 49
- problem solving scenarios
 - detecting leaks in boot process 69
 - memory hogs 68
 - memory leaks 63
 - peak usage accounting 68
- process selection

- described 21
- dialog box 21
- for Linux targets, user-space only 22, 79
- process, defined 101

R

- Real Time Process components 10
- resolution, in fragmentation map 39
- rmmod 13, 78
- routines
 - defined 101
 - memPartAlignedAllocInternal() 4
 - memPartFreeInternal() 4
- RTP
 - degraded performance 76
 - running on your target 10
 - support 11, 19
 - unresolved symbols 76
 - use restrictions on simulators 11

S

- saving, configuration file 47
- shared data region support 10
- Show Module Names option
 - in Details Viewport view 41, 44
- Show Modules option
 - in Preferences dialog box 47
- Show Offsets option
 - in Details Viewport view 41, 44
 - in Preferences dialog box 47
- snapshot
 - creating/viewing 53
 - opening 54
- starting demo program, with MemScopeDemo 18
- status messages
 - connection 72
 - overflow 73
- sticky button
 - defined 101
 - Show Modules 47

- Show Offsets 47
- symbol resolution
 - described 19
 - dfwserver (VxWorks) 19
- System Viewer
 - automatic support 60
 - integration with Memory Analyzer 7, 59
 - Memory Analyzer events 95

T

- tab views
 - Kernel Configuration, Components 10
 - Preferences - Aggregate 30, 48, 64
 - Preferences - Database 51
 - Preferences - Fragmentation 50
 - Preferences - General 41, 47
 - Preferences - Time 35, 49
 - Preferences - Tree 33, 49
- tables
 - Aggregate Allocations (Aggregate view) 27
 - Details (Time view) 35
 - Individual Allocations (Aggregate view) 30
- target connection lost
 - see troubleshooting
- target-side modules 4, 5
- target-to-host link 4, 5
 - see also architectural summary
- tasks
 - defined 101
 - in Aggregate Allocations table 28
- thread analysis
 - for Linux 2.4 kernels 22
 - for Linux 2.6 kernels 22
- Time view
 - description 34
 - Details table 35
 - elements 34
 - graphs 34
- timestamps, in Time view 35, 39
- TIPC, WDB_TIPC in an AMP environment 12
- Tree view
 - description 31
 - elements 32

- troubleshooting
 - A option 75
 - guide 74
 - Memory Analyzer GUI 80
 - target connection lost 74
 - target server 74
- tWdbTarget
 - adjusting Memory Analyzer priority 75
 - in architectural description 3
- tWdbTask
 - error, target not responding 75
 - ppc VxWorks target crash explanation 84
 - target-side module 4

U

- Unresolved Symbols view 45
- user-space processes
 - for Linux, only type supported 22, 79

V

- variables, in PATH statement 13, 78
- verbosity
 - Console and target compared 15
 - defined 101
 - Warning 15, 45
- viewing database files 55
- viewing source code
 - debugging 80
 - defined 6
 - setup 41
- views
 - Aggregate view, description 26
 - common elements 56
 - Console, description 44
 - Details Viewport, description 40
 - Fragmentation view, description 36
 - Interface Options, description 46
 - list 24
 - Time view, description 34
 - Tree view, description 31

W

WARNING

- A option usage [10](#)
- MemAgent, in patching [21](#)
- target verbosity [15, 45](#)

WDB

- ensuring WDB support for VIP [12](#)
- no support for WDB_TIPC connection [12](#)
- working in AMP environment [12](#)

Workbench

- starting Memory Analyzer [13](#)
- using dfwserver [19](#)

X

x86 target

- frame pointers must be enabled [10, 76](#)
- using debugger crashes process [83](#)