

Memory Tracker Toolkit

This document is part of a limited beta conducted by SCEA Developer Support. If you have any feedback, please file an issue on the Developer Network under the Simulation tracker.

Please note that the specifications contained in this document are preliminary and subject to change without prior notice.

© 2010 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Table of Contents

1 Overview	3
The libmtt Library.....	3
2 libmtt Integration	4
Setting up your Visual Studio Project	4
Instrumenting Memory Operations	4
3 Creating a Dataset	6
Adding System-Level Allocations	6
Capturing a Profile.....	6
Creating the Dataset	6
4 Using The Dataset Viewer	8
Menu Items.....	8
Selector Panel	8
Common Dialog Boxes	9
Views	10
5 Using the MultiViewer	16
Menu Items.....	16
Selector Panel	16
Views	16

1 Overview

The Memory Tracker Toolkit (MTT) is a set of applications and a library designed to profile memory usage and help troubleshoot common memory-related issues. Severe problems such as memory leaks, bad allocations, and heap fragmentation, along with general trends such as allocation size, lifetime, and allocator wastage can be easily tracked. By instrumenting custom allocators with logging functions or patching the executable to examine system-level functions, developers can generate logs with minimal run-time overhead that can later be compiled, read and analyzed through the Dataset Viewer applications.

The libmtt Library

libmtt is MTT's logging and instrumentation library. It contains definitions for all the custom instrumentation and logging functions. It also contains information used by the included patching utility to generate profiling information for the base-level system allocation functions.

Because MTT only works with applications that are linked with libmtt, the library is designed to be extremely lightweight. It does not add any processing overhead until you patch the application or add instrumentation functions and takes less than 200KB of executable space, so you should be able to leave it linked with your application without concern for overhead.

If you are only planning to use libmtt for a picture of the base level system allocators, simply linking your project against `libmtt.a` and running your executable through the included patcher is enough. If you need more advanced functions such as custom heap, pool, or stack allocator logging, allocation tagging, or custom events, you will need to add instrumentation. For more information on the logging functions, see the libmtt header files `libmtt_custom_allocators.h` and `libmtt_log.h`.

2 libmtt Integration

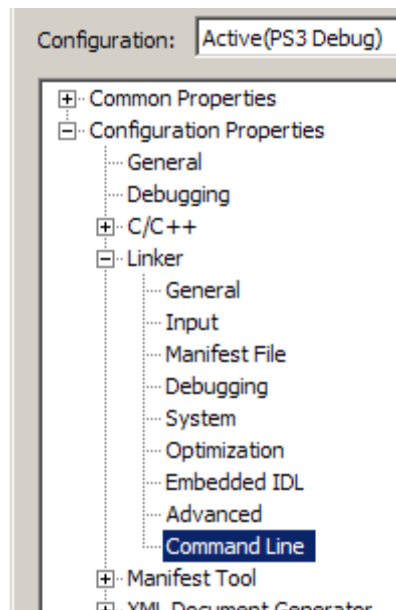
To integrate libmtt with an existing Visual Studio project, you'll need to alter your project settings as well as adding instrumentation for custom functions.

Setting up your Visual Studio Project

To setup the Visual Studio project:

- (1) In Visual Studio, open the project properties window.
- (2) Under Linker, select Command Line as shown in Figure 1.

Figure 1 Visual Studio Project Properties



- (3) Paste the following text replacing `<libmtt_dir>` with the appropriate path for your installation:

```
-Wl,--whole-archive <libmtt_dir>\libmtt.lib -Wl,--no-whole-archive
-Wl,--keep=<libmtt_dir>\libmtt_keep_list.txt
```

These options make sure that none of the instructions from libmtt used for patching are stripped as unused code.

Instrumenting Memory Operations

libmtt has two header files:

- `libmtt_log.h` is the basic header that you will use most often. It provides functions for initialization, shutdown, marking frame transitions, and inserting custom events into the log.
- `libmtt_custom_allocators.h` contains all the functions for instrumenting custom allocators; ideally it only needs to be included in the applications that define your custom allocators.

This example shows the basic steps required to instrument an application using the pool allocator, `sys_mempool`, that is part of the PlayStation®3 SDK.

To instrument your application:

- (1) Initialize libmtt.

The initialization function `mttLogInit()` takes one argument, allowing you to set the destination of the log output file. Each libmtt logging function checks to make sure that libmtt has been initialized. If it has not, the function will automatically call the initialization function with the default log location.

```
mttLogInit("/app_home/mtt_logs/sample_log.txt");
```

- (2) Register the allocator and any custom tags you plan on using.

Allocators and tags in libmtt work by associating a 64-bit unsigned integer ID, which are assigned in your application code and should be registered before use. Allocators commonly use their base memory address as an ID, while tags frequently use enumeration values. Zero is reserved as the nil value for tags. The following snippet sets up a `sys_mempool` allocator and a tag for the graphics subsystem:

```
sys_mempool_t mempool;
sys_mempool_create(&mempool, poolMemory, 65536, 8, 8);
libmtt_register_custom_pool_allocator((uint64_t) mempool, "Main Pool");

enum Tags{ NOTAG = 0, GRAPHICS = 1 };
libmtt_set_tag_name( GRAPHICS, "Graphics" );
```

- (3) Log the memory allocations and frees.

If you have access to the allocator's source, it is easiest to instrument them directly. Otherwise, you can create a wrapper function to combine the allocation and the log operation, or you can directly insert the log operations in the main application. In either case, the calls to libmtt will look something like this:

```
allocated = sys_mempool_allocate_block(mempool);
libmtt_custom_pool_alloc(mempool, allocated, 8, 0, GRAPHICS);
```

- (4) Flush the log.

libmtt uses a buffered output system to improve performance, but this means that it must be informed when the program ends so that it can flush the buffered log. In your program's shutdown routines, add a call to `libmtt_log_flush_to_disk()`.

3 Creating a Dataset

After your program has been integrated with libmtt as described in "[libmtt Integration](#)," the next step is to patch your executable to add logging to the system-level allocations .

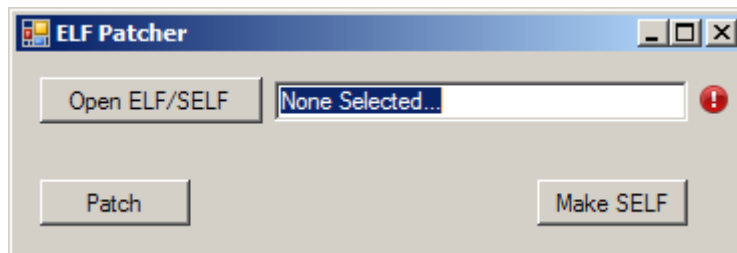
Patching System Level Allocations (Optional)

To add logging for system-level allocations:

- (1) From the ELF Patcher utility, open your `.elf` or `.self` file.

If a valid name has been entered, the red exclamation point icon disappears.

Figure 2 ELF Patcher Utility



If you select a `.self` file you will be prompted to save an intermediary `.elf` file.

- (2) Click Patch to apply the patch to the ELF.

A dialog box appears with the results of the patch. Assuming the ELF was patched successfully, dismiss the dialog box.

- (3) Click the Make SELF button to convert the patched ELF into an executable SELF file.

If you did not properly link your ELF with libmtt, or if the ELF has already been patched, an error message will be displayed.

Capturing a Profile

To capture a profile:

- (1) Run the SELF file and libmtt will automatically output a log to the location specified by `mttLogInit`, or the default:

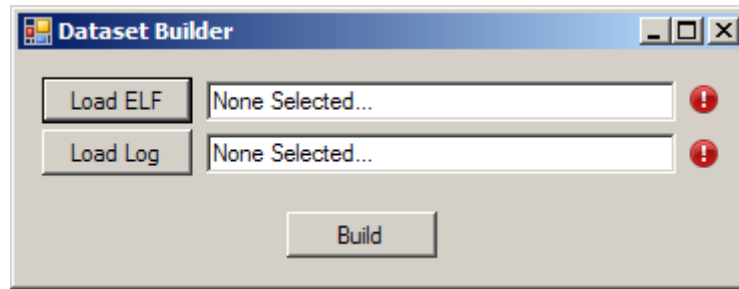
`/app_home/libmtt_log.txt.`

Before the log can be used with the Dataset Viewer however, you will have to convert it to the compressed dataset format using the Dataset Builder as described in the next section.

Creating the Dataset

To create your dataset:

- (1) In the Dataset Builder, load the ELF file (not SELF) and the libmtt generated log file.

Figure 3 Dataset Builder Load Dialog Box

- (2) Click Build and choose an output destination in the save dialog that appears.
- (3) Press OK after the status dialog shows building completed successfully.

Depending on the size of the log file and other factors, building the dataset could take several minutes. Pressing cancel at any time during the build process will abort the build and optionally delete the partial file.

Once completed, you will have a dataset ready to be used by the Dataset Viewer.

4 Using The Dataset Viewer



The Dataset Viewer is the primary means of analyzing the generated datasets. With it, you can analyze a single dataset in great detail and quickly identify potential performance or memory management issues. The Dataset Viewer is designed with the goal of helping identify the most important issues easily, and there are tools which will help you quickly track the origin of memory leaks, identify performance spikes, find the most active call stacks, and locate some of the most common types of memory mismanagement.

There are a number of controls that appear globally, used to control the entire view, including menus, controls within the Dataset Viewer Selector panel, and common dialog boxes.

Menu Items

The Dataset Viewer includes the menus and options listed in Table 1.

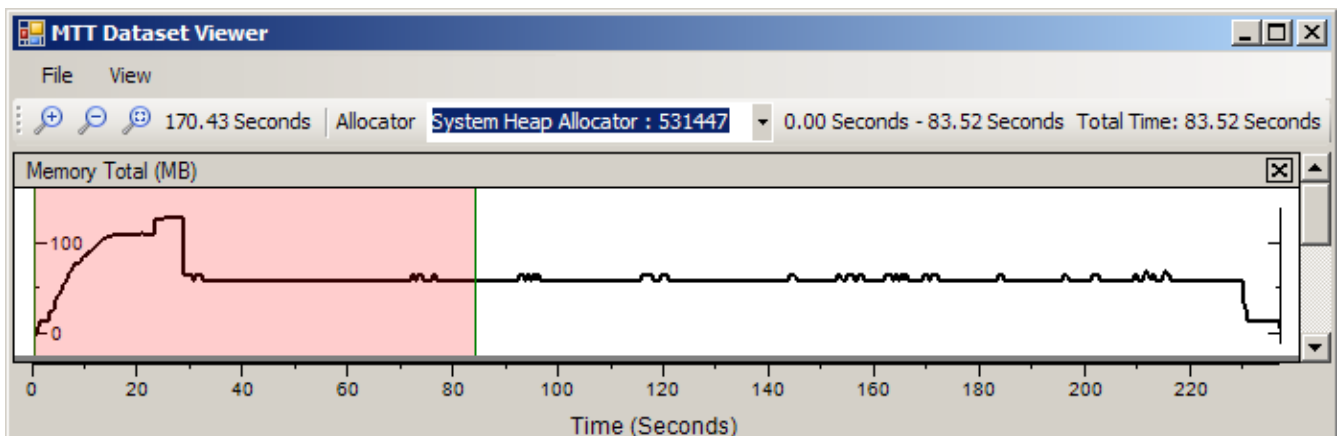
Table 1 Dataset Viewer Menus

Menu	Option	Description
File	 Open Log...	Open a dataset. Files must have the extension .ds.
View	 Select Graphs	Opens a dialog box in which you can select the graphs to display in the selection panel.

Selector Panel





The upper panel of the Dataset Viewer displays the “selector.” It is your primary means of visualizing data and choosing particular time ranges to investigate further. The highlighted region represents the range of data being visualized in the lower pane.

Figure 4 Selector Panel



The Selector panel includes the controls and status messages listed in Table 2.

Table 2 Selector Panel Controls

Icon	Control Name	Description
	Zoom	Sets the visible range to the selected range. See “Heap View” for information on special behavior.
	Zoom Out	Zooms the visible range out by a factor of 4.
	Zoom To Full	Zooms out to the entire range of the dataset.
	Time	The time represented by the pixel under the mouse cursor.

Icon	Control Name	Description
	Allocator	The name and ID of the currently displayed allocator.
	Selected Range	The start and end times of the currently selected range (highlighted).
	Graphs	A stack of various graphs of the data displayed in a scrolling list. The graph's name and the Y-Axis units are shown in the upper bar of each graph. Graphs may be resized vertically, and can also be reordered by dragging from the title bar. The selected range is highlighted in red. Change the range by dragging the start or end marker. You can also click and drag to select a range. Pressing Ctrl-A selects the entire visible range. See "Heap View" for special behavior).
	Timeline	All graphs share the same timeline, displayed at the bottom of the panel.

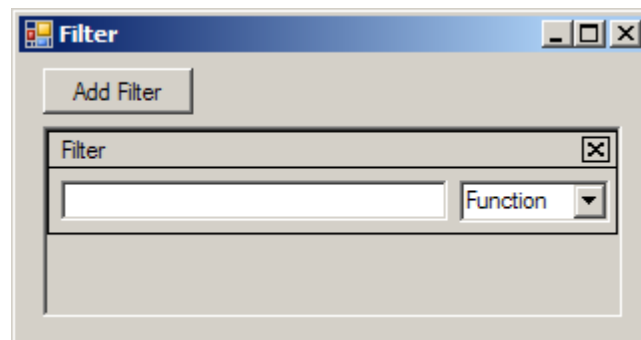
Common Dialog Boxes

The Dataset Viewer includes dialog boxes to set filters and to view the content of the call stack. They are available in many contexts.

Add Filter

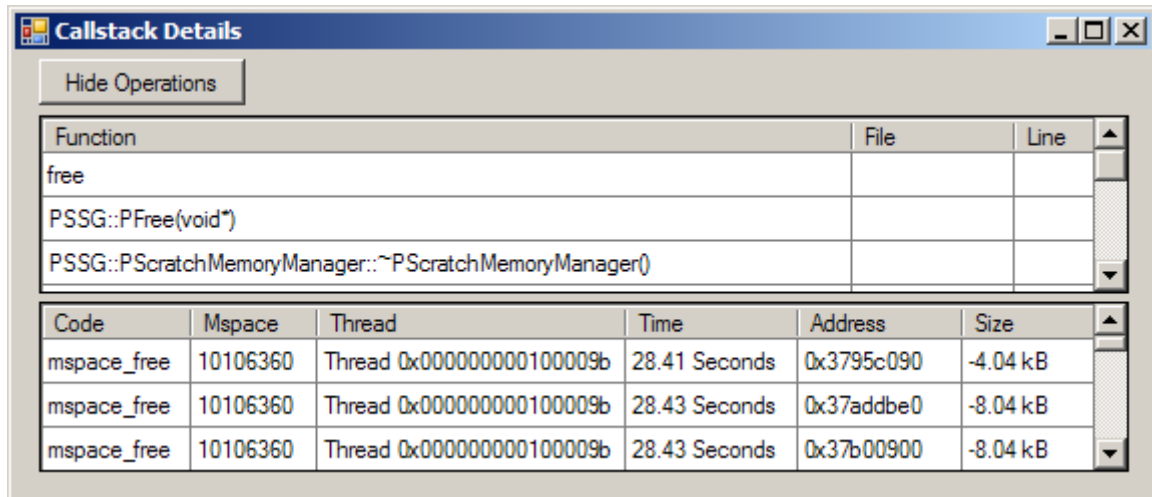
You can filter the content shown in many Dataset Viewer views by clicking the Add Filter (🔍) button to display the Filter dialog box. In the dialog box, click the Add Filter button to add an arbitrary number of filters. The drop-down box on the right of each filter selects the field to which the filter is applied. Closing the dialog box applies the filters to the current view.

Figure 5 Filter Dialog Box



Call Stack Details

In any of the Dataset Viewer views with call stacks as the primary column, double clicking an entry brings up the Callstack Details dialog box. The upper panel displays the complete call stack. The lower panel shows information on each individual operation involved. Click the Show/Hide Operations button to toggle the lower panel.

Figure 6 Call Stack Inspection Dialog Box

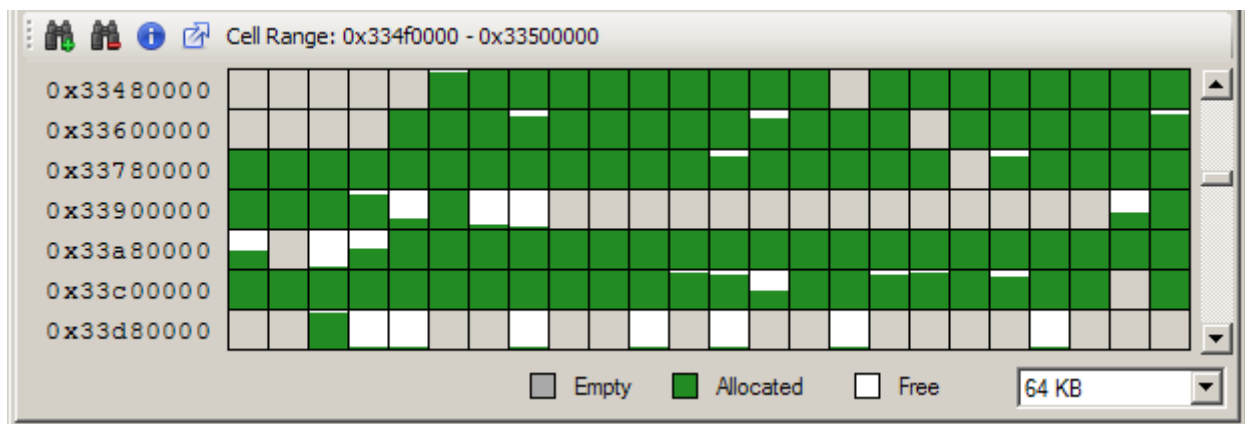
Views

Views are displayed in the lower panel of the Dataset Viewer below the Selector panel. They display different kinds of information related to the range selected in the Selector panel. Choose which view to display using the tabs along the top of the lower panel.

Heap View

The Heap view provides a complete view of memory at a given time. When the Heap view is displayed, the selector's starting point is fixed at 0. As a result:


- Zooming out always centers on the end of the selected range and zoom by a factor of 4.
- The Heap view's selected range is independent from the other views.

Figure 7 Heap View

The Heap view includes the controls and status messages listed in Table 3.

Table 3 Heap View Controls

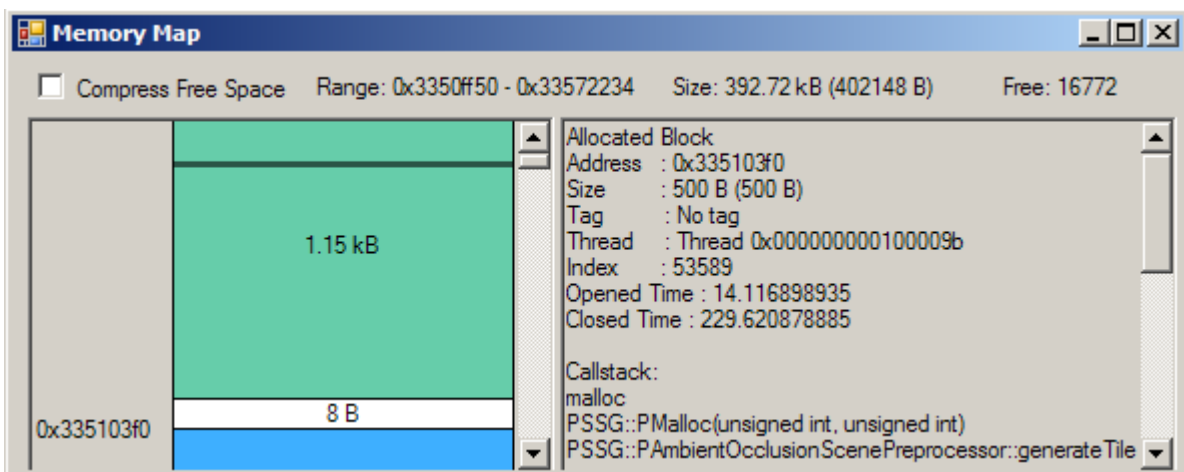
Icon	Control Name	Description
	Zoom Heap	Fit the block size as close as possible to fill the view with the selected range.
	Zoom Out	Zooms out by a factor of 2.
	Get Information	Displays a popup window very similar to the Callstack View where you

Icon	Control Name	Description
		can inspect the call stacks responsible for altering the selected region of memory.
	Show Memory Map	Brings up the Memory Map dialog box.
	Cell Range	The address range of the last cell the mouse hovered over.
	Block Size	The drop-down menu in the lower right corner of the view displays the current size represented by each block in the view. Select another value from the menu to change the size of each block.

Memory Map Dialog Box

Clicking the Memory Map button in the Heap view displays a detailed map of the memory range selected in the Heap view. Allocated blocks are displayed in colors that indicate the block's tag; free blocks are displayed in white. Hovering over blocks shows the start and end addresses for the block to the left and detailed information in the panel to the right. If the Compress Free Space option is checked, all empty blocks will be displayed with the same height. If the selected range slices a memory block, the full block will be shown with a dark dividing line showing the start of the selected range.

Figure 8 Memory Map Dialog Box



Statistics View


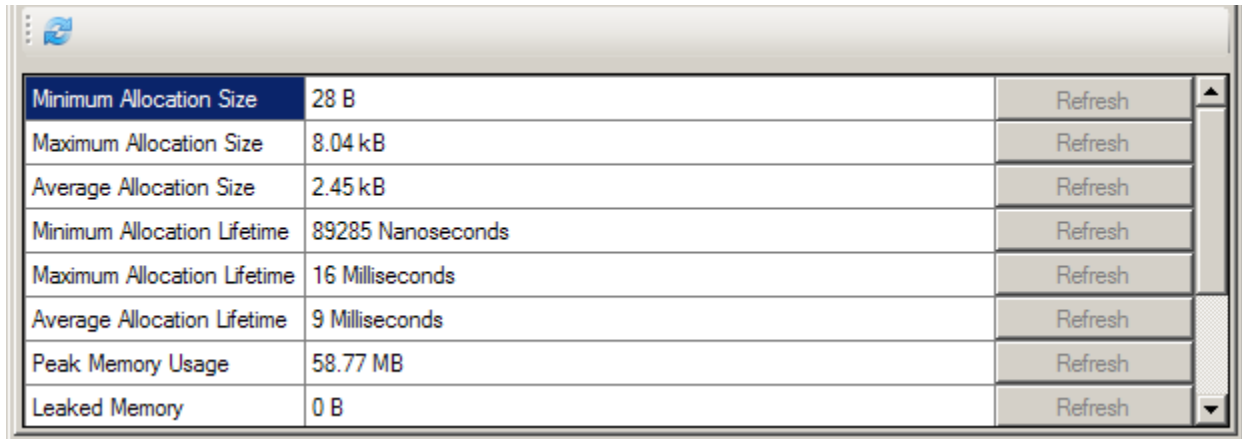

The information displayed in the Statistics view is designed to help you identify performance bottlenecks and patterns. Most statistics are refreshed in sync with selecting the range; however to avoid delays caused by calculating certain expensive statistics, some require a manual refresh. To update a statistic in the view, click the associated Refresh button. You can also click the Refresh All () button to update all statistics in the view after changing the range selection.

Figure 9 Statistics View


Minimum Allocation Size	28 B	Refresh
Maximum Allocation Size	8.04 kB	Refresh
Average Allocation Size	2.45 kB	Refresh
Minimum Allocation Lifetime	89285 Nanoseconds	Refresh
Maximum Allocation Lifetime	16 Milliseconds	Refresh
Average Allocation Lifetime	9 Milliseconds	Refresh
Peak Memory Usage	58.77 MB	Refresh
Leaked Memory	0 B	Refresh

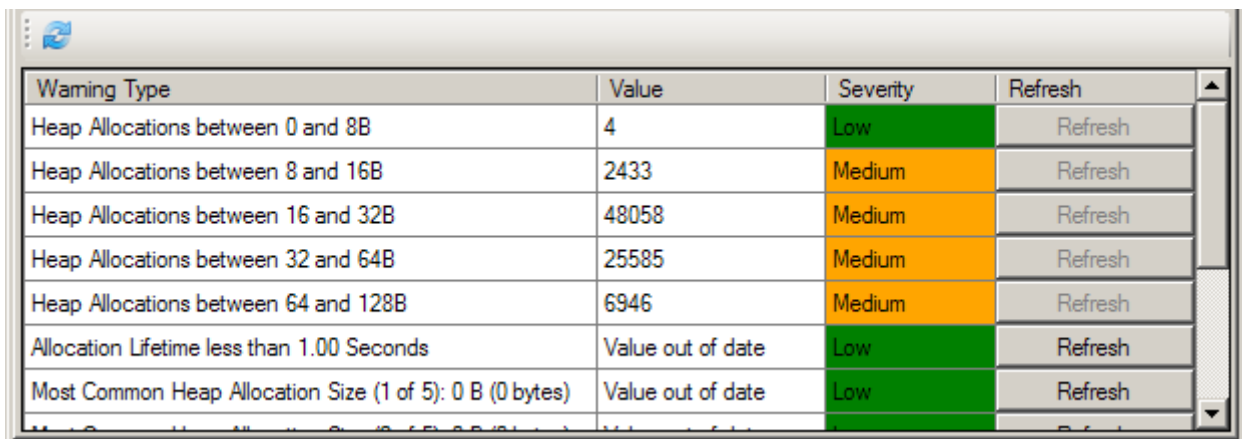
The Statistics view includes the control listed in Table 4.

Table 4 Statistics View Control

Icon	Control Name	Description
	Refresh All	Refreshes all items requiring an update.

Warnings View


The Warnings view, like the Statistics view, displays aggregated information about the selected range. The key difference is that each item in this view displays information designed to help identify a specific issue as opposed to the Statistics view's more general profiling. In the severity column, you can see a relative indication of how much the issue is likely to affect your application. As in the Statistics view, you can double click an item to bring up a dialog box with additional information, including the offending call stack and fix suggestions.

Figure 10 Warnings View


Warning Type	Value	Severity	Refresh
Heap Allocations between 0 and 8B	4	Low	Refresh
Heap Allocations between 8 and 16B	2433	Medium	Refresh
Heap Allocations between 16 and 32B	48058	Medium	Refresh
Heap Allocations between 32 and 64B	25585	Medium	Refresh
Heap Allocations between 64 and 128B	6946	Medium	Refresh
Allocation Lifetime less than 1.00 Seconds	Value out of date	Low	Refresh
Most Common Heap Allocation Size (1 of 5): 0 B (0 bytes)	Value out of date	Low	Refresh

The Warnings view includes the control listed in Table 5.

Table 5 Warnings View Control

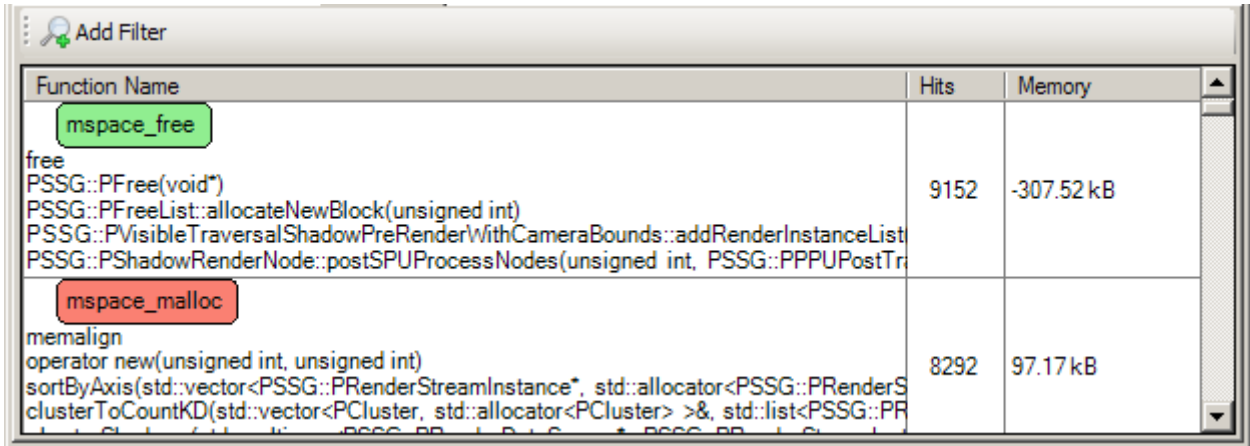
Icon	Control Name	Description
	Refresh All	Refreshes all items requiring an update.

Call Stack View

The Call Stack view is used to identify the stacks responsible for allocating or freeing the majority of memory in your application. Stacks can be sorted by either total memory effect or number of calls. The first five lines of each call stack are displayed in this view; to explore the full entry do one of the following:

- Double click to open the Callstack Detail dialog box.
- Hover over an entry to display a tooltip with the complete call stack.

Figure 11 Call Stack View



The Call Stack view includes the control listed in Table 6.

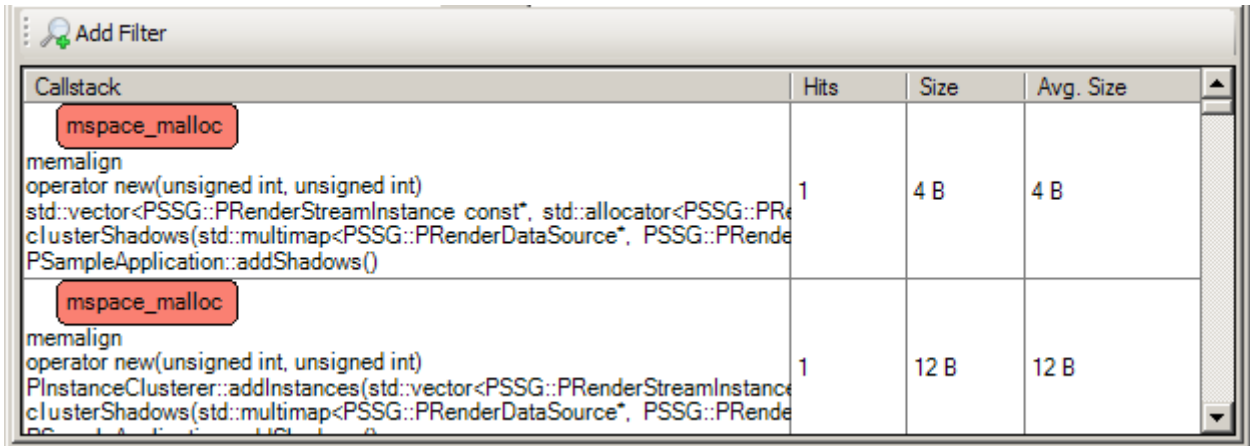
Table 6 Call Stack View Control

Icon	Control Name	Description
	Add Filter	Displays the Add Filter dialog box.

Leaks View

The Leaks view displays all call stacks in the selected range that leak memory. A “leak” is defined as any memory allocation that happens within the selected range, but is not freed by the end of it. The view can be sorted by hits, total size leaked, or average amount leaked. Double click on any of the rows to display the [Call Stack Details](#) dialog box.

Figure 12 Leaks View



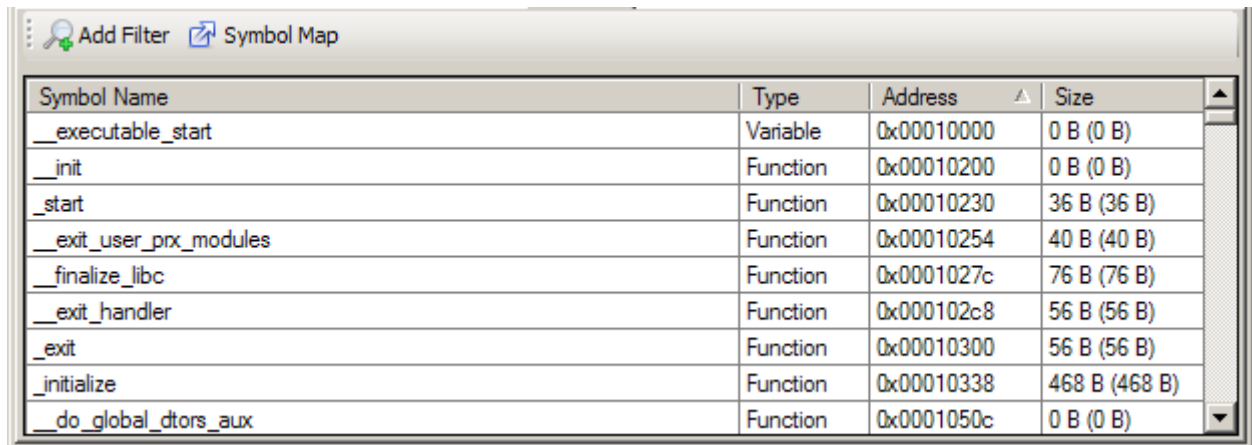
The Leaks view includes the control listed in Table 7.



Table 7 Leaks View Control

Icon	Control Name	Description
	Add Filter	Displays the Add Filter dialog box.

Symbols View



The Symbols view allows you to inspect the symbol table of the elf. This view shows the actual memory size used by a class or set of functions in your elf. You can also examine the data layout to ensure locality and help improve cache hits.

Figure 13 Symbols View


	Add Filter		Symbol Map
Symbol Name	Type	Address	Size
__executable_start	Variable	0x00010000	0 B (0 B)
__init	Function	0x00010200	0 B (0 B)
__start	Function	0x00010230	36 B (36 B)
__exit_user_prx_modules	Function	0x00010254	40 B (40 B)
__finalize_libc	Function	0x0001027c	76 B (76 B)
__exit_handler	Function	0x000102c8	56 B (56 B)
__exit	Function	0x00010300	56 B (56 B)
__initialize	Function	0x00010338	468 B (468 B)
__do_global_dtors_aux	Function	0x0001050c	0 B (0 B)

The Symbols view includes the controls listed in Table 8.

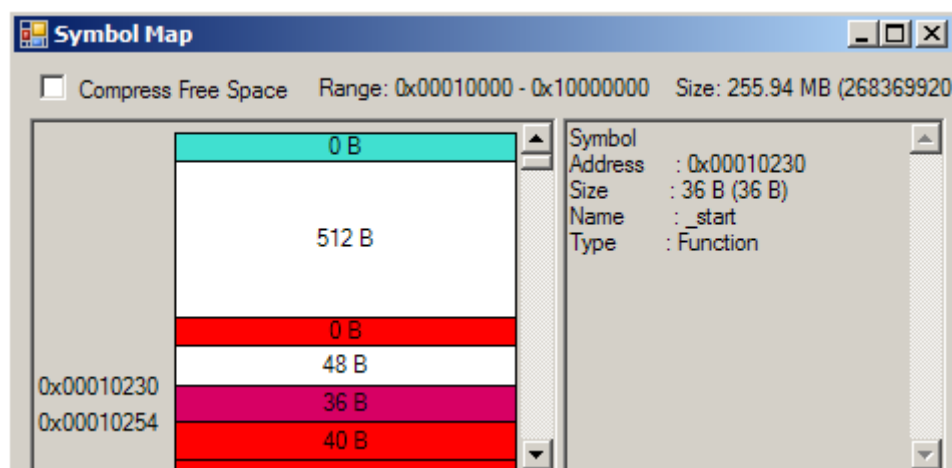
Table 8 Symbols View Control

Icon	Control Name	Description
	Add Filter	Displays the Add Filter dialog box.
	Symbol Map	Displays the Symbol Map dialog box.

Symbol Map Dialog Box

The Symbol Map displays a visual representation of the memory layout of the symbol table, with filtered symbols omitted and rendered as free space. Hovering over a block displays the start and end addresses of the block on the left, and detailed information on the right. If the Compress Free Space option is checked, all blocks of free space are shown with the same height, though the labels still reflect their actual size.

Figure 14 Symbol Map Dialog Box






5 Using the MultiViewer

The MultiViewer is designed as a companion tool to the Dataset Viewer, allowing you to visualize the aggregate information of the Statistics and Warning Views from multiple allocators across multiple datasets simultaneously. As a higher level tool it is meant to help teams track the evolution of their application's memory characteristics. The interface is nearly identical to the Dataset Viewer, so it should feel fairly familiar.

Menu Items

The MultiViewer includes the menus and options listed in Table 9.

Table 9 MultiViewer Menus

Menu	Option	Description
File	 Open Datasets...	Open datasets, discarding those currently loaded. Files must have the extension .ds.
	 Add Dataset...	Open datasets, keeping those currently loaded. Files must have the extension .ds.
View	 Select Graphs	Opens a dialog where you can select which graphs will appear in the selection pane.

Selector Panel

The Selector panel in the MultiViewer has the same behavior as the Selector panel in the Dataset Viewer, as described in "[Selector Panel](#)."

Views

Comparison View

The Comparison view shown in Figure 15 displays all the various statistics across all loaded datasets. Each data column represents the dataset's allocator. The columns appear in the same order that they are charted on the graphs in the Selector panel.

The label of each column is formatted as:

```
<Filename>:<Allocator Index>
```

You can drag the columns to reorder them, and the graphs will update appropriately. You can also right click on a column to display a context menu where you can delete a graph.

Figure 15 Comparison View

Comparison					
Statistics	DT_CustomEvent.ds:1	DT_CustomEvent.ds:	simple.ds:0	DT_CustomEvent.d	DT_CustomEvent
Average Lifetime	0.00 Seconds	23 Milliseconds	104022 Nano...	31 Milliseconds	36 Milliseconds
Minimum Size	0 B	8 B	256 B	16 B	32 B
Maximum Size	0 B	8 B	256 B	16 B	512.00 kB
Average Size	0 B	8 B	256 B	16 B	1.15 kB
Remaining Memory	0 B	0 B	0 B	0 B	0 B
Peak Memory	0 B	4.00 kB	64.75 kB	8.00 kB	656.00 kB
Short Allocations	0	0	0	0	0
Leaked Memory	0 B	0 B	64.00 kB	0 B	64.00 kB
Allocations less than 8B	0	0	0	0	0
Allocations between 8B and 16B	0	512	0	0	0