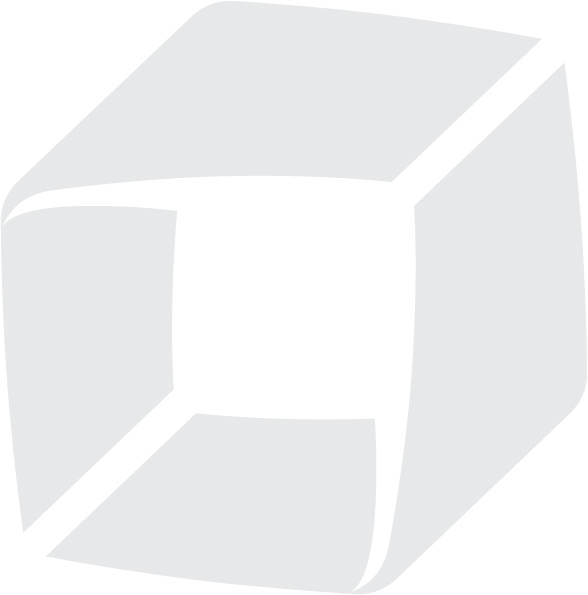
­

Dynatrace Advanced Training Reference

Updated <Date>



# OVerview

This document details the features and functions that are covered in the Dynatrace Advanced Training. The aim of this training is to enable users to deep dive into Application Performance and find the root cause of application problems.

Table of Contents

* Session Storage 3
  + Storing Sessions 3
  + Store Sessions for a Specific Timeframe3
  + Store Specific PurePaths (recommended method)4
  + Exporting Stored Sessions5
  + Importing Sessions5
* Memory Diagnostics 6
  + Total Memory6
  + Create a Total Memory Snapshot7
  + Snapshot Overview Tab7
  + Hot Spots8
  + Other Tabs of Interest9
  + Comparing Memory Snapshots9
  + Analysis Drill-Down9
  + Selective Memory9
  + Selective Memory Metrics10
  + Group by Allocation Time10
  + Create Memory Rules from Selective Memory Content11
  + Compare Selective Memory Snapshots11
* Thread Diagnostics 13
  + Overview13
  + Grouping Threads13
  + Comparing Thread Dumps 15
  + Limitations16
  + Overview Tab16
  + Thread Performance Metrics 16
  + Stack Trace17
  + Owned Monitors 17

# Session Storage

In Geico’s implementation of Dynatrace, all transactions are automatically stored on disk. The limitation of storing all transactions is disk space and Dynatrace only stores up to 2 weeks of transactions by default. Due to the level of traffic and disk limitations at Geico, all transactions are stored for approximately 4-5 days. If additional time is needed to triage with these transactions, the session may be stored and exported to external storage.

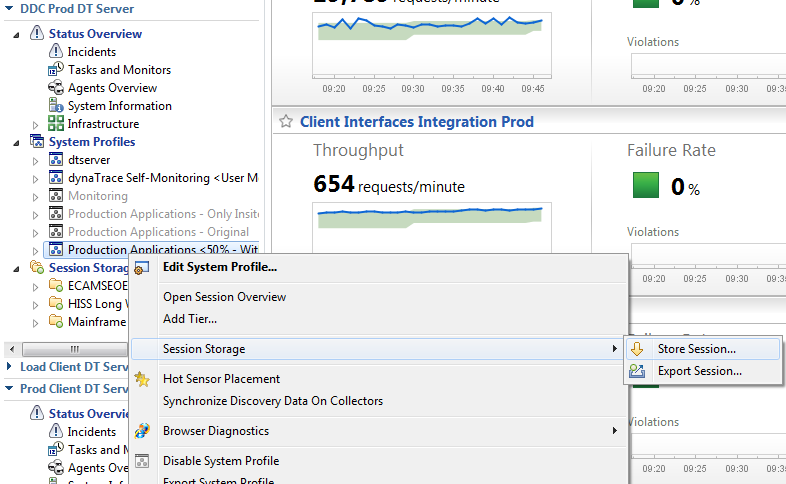
Note: Metrics are not deleted like transactions, but they are aggregated over time and can be analyzed over large time frames.

Storing Sessions:

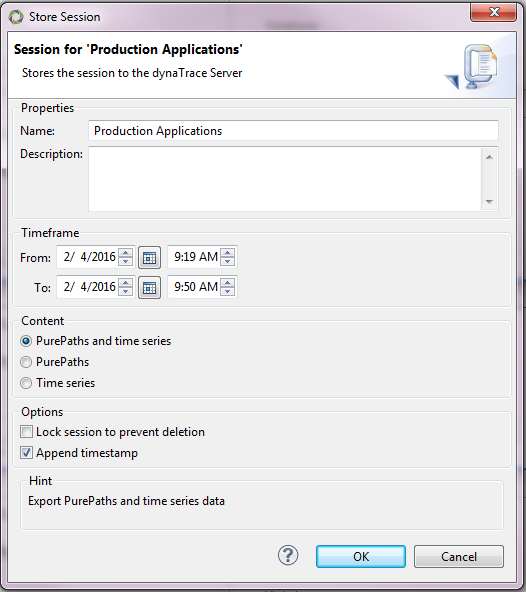
There are 2 ways to store sessions in Geico’s implementation of Dynatrace. The first way captures all PurePath and PureStack info for the system profile during the specified time frame. While this ensures all data is captured, the files can be quite large. The second way is to select just the PurePaths you would like to store *(PREFERED METHOD).* Both ways are detailed below.

Store Sessions for a Specific Timeframe:

* From the Cockpit list, right-click on the target system profile.
* Click Session Store, then Store Session.

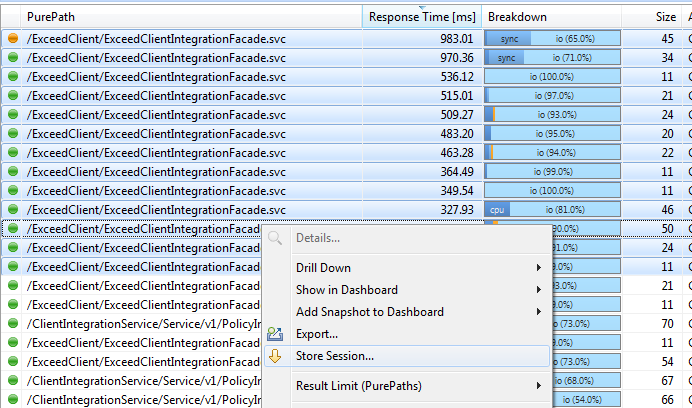


* The following dialog box will appear enabling you specify a timeframe, label, and options to lock to prevent deletion and append a timestamp.



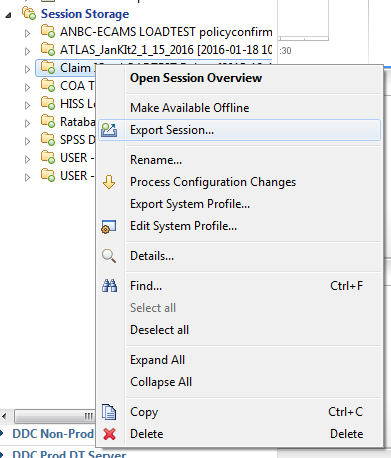
Store Specific PurePaths:

* Shift-Click the PurePaths you would like to store, right click, and select Store Session.
* The Store Session Dialog will appear like in the previous example, but without a timeframe as you’ve selected the specific PurePaths you’d like to store.



Exporting Stored Sessions:

* Once a session is stored, it will be available underneath the respective system profile and can be exported to external storage.
* From the Cock put, open Session Storage
* Right click on the desired stored session and select Make Available Offline. This ensures all packages are included that allow the session to be viewed in a Dynatrace Client without a Dynatrace Server.
  + Simply hitting Export will allow you to save the session externally, but you will have to import it onto a Dynatrace Server to view it.



* When prompted by the export dialog box, navigate to where the session will be saved and specify a file name.
* Click export to create a compressed file with the .dts extension which can be viewed on any Dynatrace Client.

Importing Sessions:

* Click the Tools menu at the top of the screen.
* Select Import Session and browse to the desired session file. Once imported, the file will be available in a table in the Imported Sessions Dashlet.
* Right-Clicking the imported session file provides a list of options for analyzing and manipulating the file.

# MEMORY DiAGNostics

Dynatrace provides sophisticated memory analysis through both Selective Memory and Total Memory dumps. The following sections explain how to utilize the memory tools in Dynatrace. For an in-depth guide on how to fully diagnose memory issues and best practices, please see the Triage Best Practices Documentation. Memory dumps are post processed on the memory analysis server which is shared amongst Dynatrace servers. Please contact your Dynatrace Admin for post processing.

Caution: The following information can lead to memory dumps which will suspend the application. Please read all Dynatrace dialog boxes carefully to ensure you are aware of the impact on the application. Please exercise extra care when utilizing these features, especially in Production environments.

Total Memory:

* The Total Memory dashlet provides snapshot information about currently allocated object regardless of their connection to recorded PurePaths or configured memory rules. Total memory snapshots belong to one of the following 2 Types:
  + Trending Snapshot: These contain information about classes, instance counts, and shallow sizes.

Caution: Memory Consumption Trending snapshots are less impacting than Deep Leak-Analysis, but they still will suspend the application several seconds for each snapshot that is taken. For trending, snapshots are taken periodically.

* + Deep Leak-Analysis Snapshot: The contain information about classes, the individual instances, references between individual instances GC Size.

Caution: Deep Leak- Analysis may take take several minutes and the application will be suspended in the meantime.

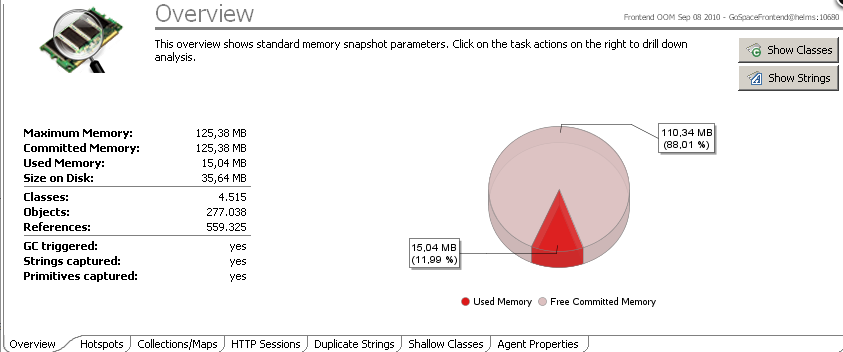
* To create a snapshot, click **Create Memory Snapshot** in the lower pane. In the trending screenshot, memory usage falls a bit after the second auto-shot. This is because the user selected **Create Memory Snapshot** > **General Options** > **Force Garbage Collection**.
* The upper pane of the dashlet displays all snapshots of the corresponding system-profile, grouped by Agents. By default, the table contains the following information for each snapshot:
  + Agent/Dump: Contains the name of the snapshot (it's creation time by default). Right-click and select Edit to change the snapshot name.
  + Labels: Snapshots can have custom labels assigned. To attach information to a snapshot, right-click and select Edit to assign or delete labels.
  + Locking State: Determines if you can delete the snapshot automatically if the server is running low on disk-space. Right-click and select Edit to change the locking state.
  + Type: The type of the memory snapshot, either Leak Analysis or Trending.
  + Status: The state of the snapshot, Finished, Finished (raw) or Not completed. A finished snapshot is ready for analysis, a raw-snapshot needs post-processing before you can analyze. A Not completed status indicates problems you discover while you create or post-process the memory snapshot.
  + Used Memory: The amount of allocated memory at the time the snapshot is created.

Create a Total Memory Snapshot:

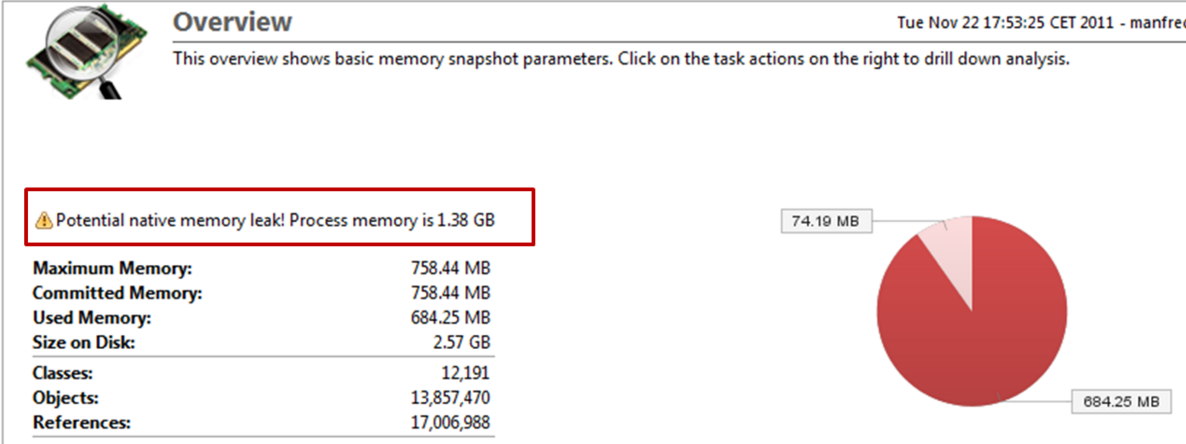
* The dialog for creating a memory snapshot allows one to determine if a single Leak-Analysis snapshot or one or multiple trending snapshots should be created. Click Pick-Agent to select the Agent for which a snapshot should be created. Only currently connected agents are listed. You should select a single agent-instance (no agent-group or agent-mapping). At the bottom of the dialog, you can deactivate a default triggered Garbage Collector. By default, the snapshot is automatically post-processed. You can disable this in the dialog.
* If you enable Capture values of String objects, Dynatrace performs an automatic string duplicate analysis while it post-processes. If you capture the values of string, the time required to create the snapshot increases because string values have to be transferred. This option works only with Java VMs higher or equal to 6.0, and CLRs higher or equal 2.0.
* If the option Capture values of primitive fields is enabled, the values of primitives can be inspected in the Fields and Values Dialog for each instance. This option is only available for Java VMs higher or equal 6.0.

Snapshot Overview Tab:

* If an existing snapshot (which is already post-processed) is selected in the upper pane, overview information of the corresponding snapshot is presented in the individual tabs in the lower pane as depicted in the following figures. When selecting a non-post-processed snapshot, a button in the lower-pane allows to post-process it.
* This tab shows general information regarding the Heap usage at the time the snapshot was created, including maximum Heap size, currently used and committed memory, the number of Classes and (in case of an Leak-Analysis snapshot) the number of Objects and References.
* By clicking on the Show Classes Button, the content of the Memory Snapshot (all classes and instances) is opened. By clicking on the Show Strings Button all Strings contained in the snapshot are shown (only available if enabled while snapshot creation).



* In case the overview contains a message that a potential native memory leak has been detected as depicted in the image below, the process-memory of the application is higher than 120 % of the maximum heap size of your VM. Please note that this does not necessarily mean that there is a problem with native memory, it just highlights the possibility of such a leak. Native memory leaks often occur when UI resources are not closed correctly or when a problem with native buffers exists.

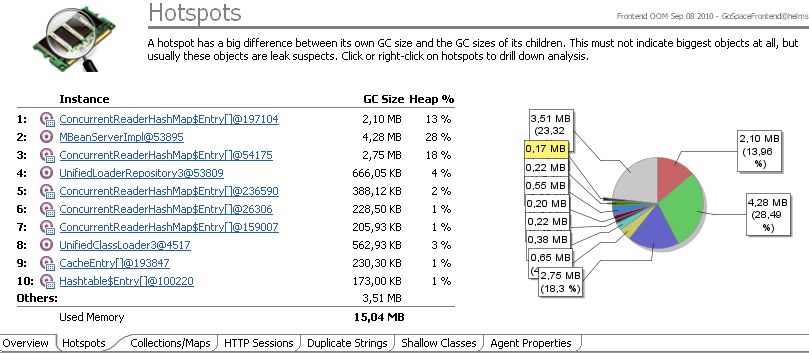


Hot Spots:

* The Hot Spots tab shows the potentially top interesting objects in the Memory Snapshot. A Hot Spot is an object that's difference between its own GC size and the maximum GC size of its child is high. A Hot Spot thus 'accumulates' a lot of other objects.

Note: Hot Spots are not ordered by their GC size but by the difference of their size to maximum size of an objects children.

* Left-click a Hot Spot to open the Shortest Path to Root By Keep Alive in a new dashlet, which displays how it is kept alive. For further analysis of a Hot Spot (e.g. analyzing the objects it keeps alive), right-click on it and select the corresponding drilldown.



Other Tabs of Interest:

* Top Collections/Maps: The Top Collections/Maps tab shows the largest (by GC size) objects on the Heap that implement the Collection or Map interfaces. Collections and Maps are shown in a separate tab since they are very frequently the root-cause of a memory leak. Similar to the other tabs, use the context-menu for further analysis of the individual objects.
* Top Http-Sessions: This tab contains the largest (by GC size) HttpSession objects on the Heap. As for the other tabs, the context-menu allows to drill-down to further analysis Dashlets.
* Top Duplicate Strings: If capturing of Strings is enabled while snapshot creation, this tab shows the top duplicate Strings, ordered by their saving potential by String internalization. The saving potential is calculated by the size of a string multiplied by its duplicate-count. Use the context-menu to drill-down to further analysis Dashlets.

Comparing Memory Snapshots:

* When selecting multiple memory snapshots, the overview will present a Chart that shows Heap statistics over time as depicted below. Click on the compare - button to compare the selected snapshots. Details can found in Total Memory Comparison.

Analysis Drill-Down:

* Within a Leak-Analysis snapshot the following drill-down analysis Dashlets are available for individual objects and usually also aggregated for classes. An aggregated analysis means that the corresponding analysis is done for multiple (or all) instances of a class. Use the context-menu on a class or instance for the following analysis Dashlets:
  + Follow References - Direct
  + Follow References - By Keep Alive
  + Shortest Root Paths - Direct
  + Shortest Root Paths - By Keep Alive
  + Keep Alive Set
  + Fields and Values

Selective Memory:

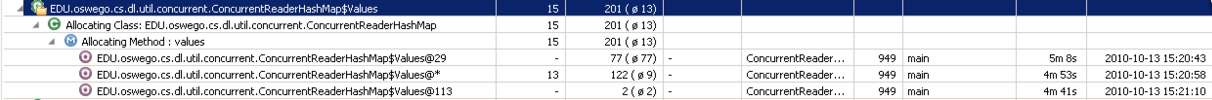
* Use the Selective Memory dashlet to create or open Selective Memory snapshots and analyze the instances in the snapshot. A selective memory dump uses memory sensor rules for instance-allocation book keeping. You get a list with all allocated instances for which you have created memory sensor rules.
* The Selective Memory dashlet shows all existing Selective Memory snapshots. You can create a new snapshot if you select an Agent, or nothing, in the upper table of the dashlet. If you select an existing snapshot, an overview of the snapshot appears. If you select multiple snapshots, a chart appears that compares the overview data.

Selective Memory Metrics:

* Class/Instances: Name of class that is monitored for instantiations. The map/collection sensor includes all classes that implement the corresponding interfaces. When you expand the class, the individual instances are grouped by their allocating class, allocating method, allocating line, and their time of allocation.
* Instances: Number of monitored class instances allocated by one method of one allocating class.
* Sum Collection Size For collections or maps, this represents the sum of the collection.size() / map.size() values of all Instances. It also contains an average of the size of all Instances of this class.
* Allocating Class: Class that allocates the instance. Hidden by default.
* Allocating Method: Method that allocates the instance. Hidden by default.
* Domain: The Appdomain (only for .NET)
* File: Name of the source file of the allocating class, if available.
* Line: Line number within the source file were the allocation is performed. If allocation takes place in a native method the line number is set to "-"
* Allocating Thread: Name of the thread that allocated the instance.
* Object Age: Age of the instance at the time the snapshot is created. Use this to find very long living instances.
* Allocation Time: When the Instance is allocated.

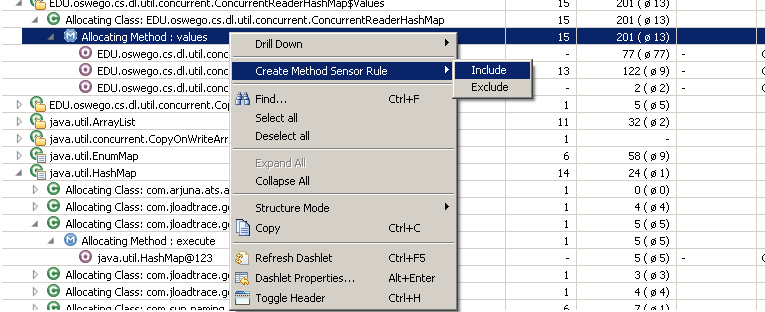
Group by Allocation Time:

* Dynatrace groups instances by allocating class, allocation method, allocating line and their time of allocation. If the same class/method/line, within the same second, allocates multiple instances of a class, Dynatrace groups them into a single node to improve visibility. The following figure shows three ConcurrentReaderHashMap$Values nodes. The first shows the allocation of a single entry. The second shows the allocation of 13 entries within a single second. The third shows the allocation of a single instance.



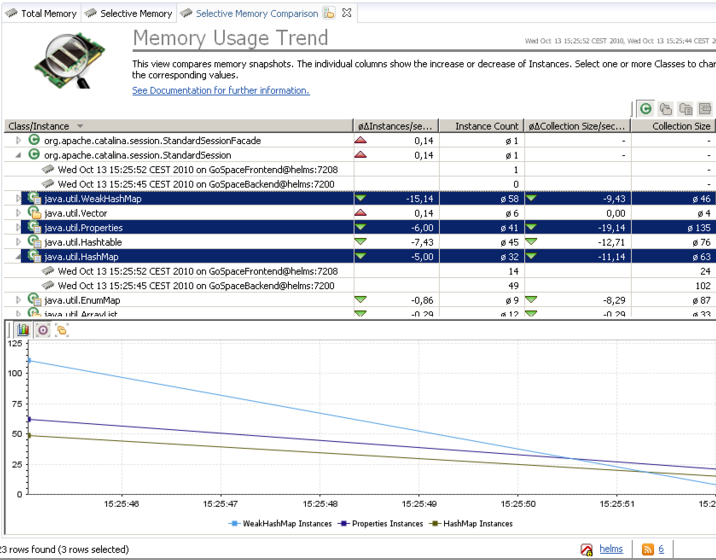
Create Memory Rules from Selective Memory Content:

* Use the Selective Memory Content dashlet to create include or exclude method rules to allocate methods. When you create an include rule, the corresponding method that allocates an instance is included into PurePaths. To create method rules, use the context-menu on an allocating method node as shown in the figure below.



Compare Selective Memory Snapshots:

* To open the Selective Snapshot Compare dashlet, select multiple snapshots and click the compare button in the overview (or use the context-menu). This dashlet compares the instance counts and collections/maps classes size of the selected snapshots. You cannot compare individual collection/map instances by size. All instances of a specific class must be aggregated.



* The view includes the following columns:
  + Class/Instance: Class that is compared. Expand to see the individual compared snapshots.
  + Average Delta Instances per second: Average change of number instances per second for this class.
  + Instance Count: Average number of instances over all snapshots for first-level entries (classes). Number of instances in this snapshot for second-level entries (snapshots).
  + Average Delta Collection size: Average change of the sum of the sizes of all instances of a class for collections/maps only.
  + Collection size: Sum of the size of all instances of that class for collections/maps only.

# Thread DiAGNostics

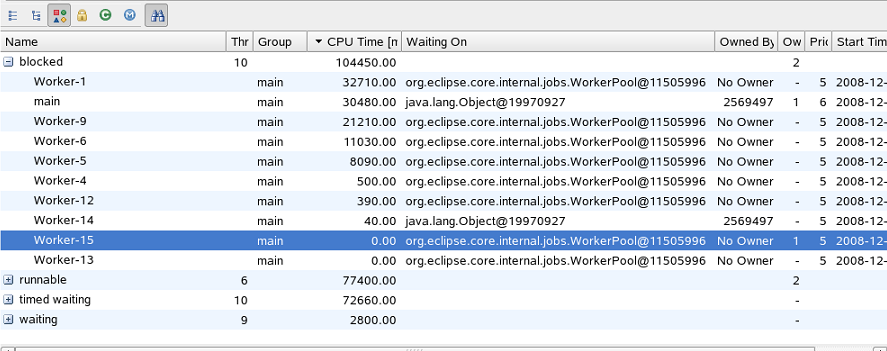
Dynatrace provides sophisticated thread analysis. Please consult with the application owner before taking a thread dump and please refer to the Triage Best Practices for details on how to best analyze thread issues.

Overview:

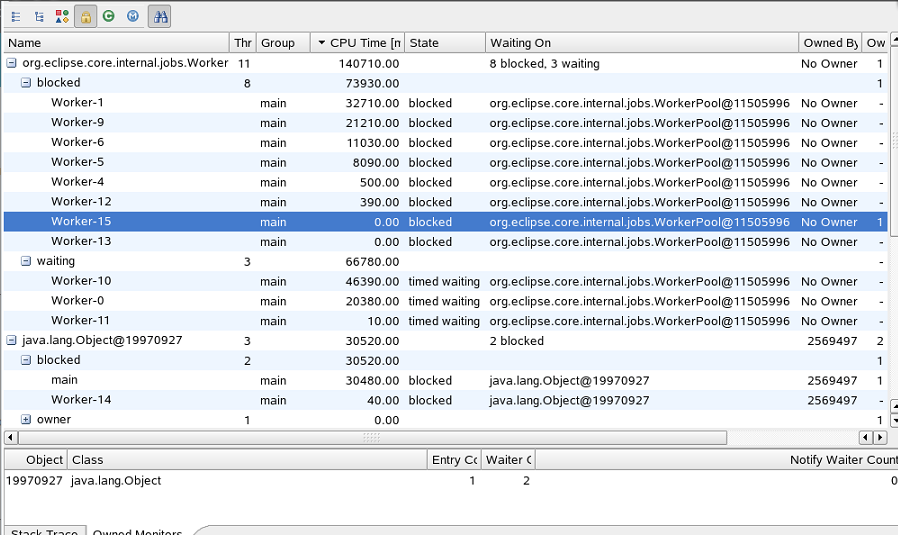
* Thread dumps provide a snapshot of all JVM/CLR threads. They are a powerful way of finding deadlocks, idle or busy thread pools, thread leaks, and more.
* There are many advantages of thread dumps compared to JVM dumps:
  + CPU time information is available.
  + Multiple thread dumps can be compared.
  + You can group threads by significant criteria.
  + You can search for PurePaths where a specific thread is involved.
  + To perform a thread dump, it is not necessary to start the VM in the console or redirect the output.
  + Thread dumps can be triggered automatically.
* While the thread information is collected, all threads are suspended to guarantee consistency of stack traces, states, and monitors.
* Thread dumps can be persisted, searched, and grouped. You can schedule thread dumps. For example, you might trigger thread dumps at times when the load on the system under diagnosis is low, to ensure that there are no threads leaking, or to compare the CPU usage of different threads to previously created thread dumps.

Grouping Threads:

* It is often tedious to analyze a traditional JVM dump because the vital information is scattered in the log file or tool. For example, you have to know which thread owns which monitor or which threads are waiting for a buffer.
* The ability to group threads by different criteria is very handy, especially in production scenarios with many concurrent threads. For example, threads grouped by their thread group show which thread group consumed the most processor time, because the CPU times are accumulated. Grouping by classes or methods is useful to see what is happening in an application at the time of the dump.
* Thread state and thread group are the most common groupings. The following figure shows a thread dump taken from Eclipse under load.

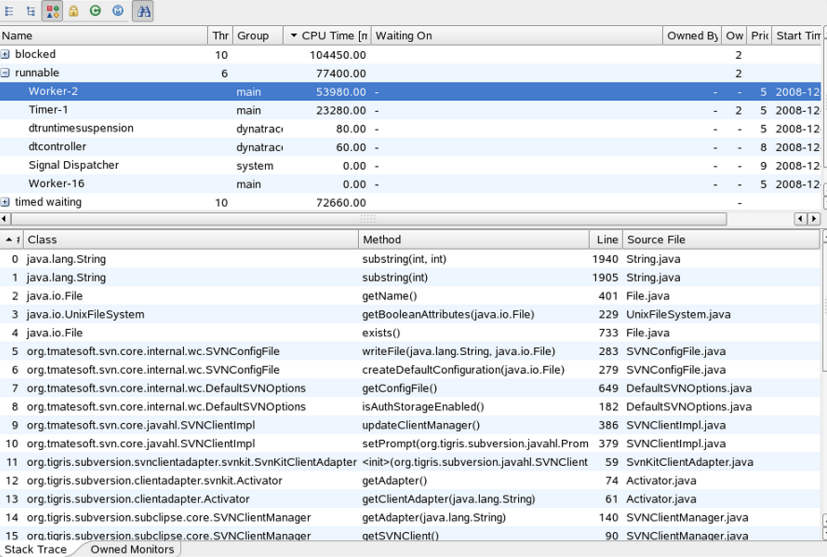


* There are 10 blocked threads and some of them have locks on monitors. Having multiple monitors locked isn't a problem in itself, but it could indicate architectural flaws. For example, it may be that the method was not intended to be called from another method that synchronized on the owned lock.
* This figure shows that two threads are waiting for this monitor. It is unclear how long they have been waiting, but it is advisable to keep an eye on them. The methods are candidates for placing additional Sensors.

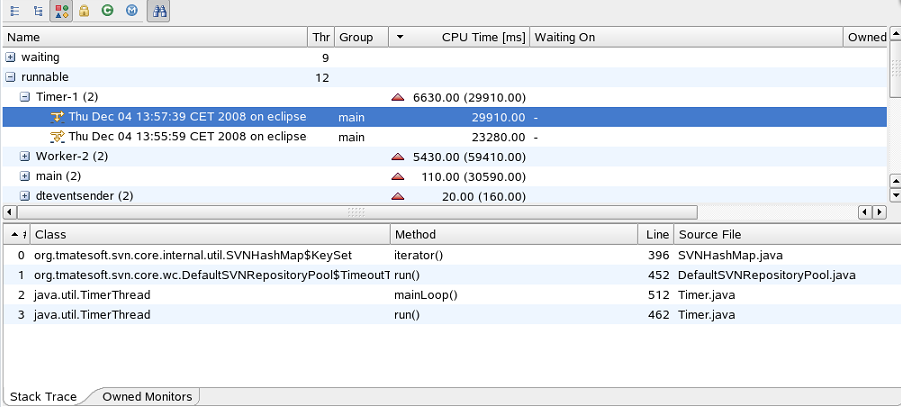


Comparing Thread Dumps:

* When the system under diagnosis appears to have random threading issues, it helps to examine all running thread.



* The most active thread is Worker-2. It is updating the Subversion state of all project files. To find out what's taking so long, another thread dump is requested, as shown in the following figure.



* The dump shows that Worker-2 was not the only busy thread. Timer-1 also consumed a lot of CPU time, probably with Subversion. To learn more, you could place additional Sensor rules for the package org.tmatesoft.svn.core. The filter feature shows all threads related to Subversion.
* Threads are compared by their ID. Because the ID changes after restarting the system under diagnosis, threads from different runs can only be compared by name. The thread name does not have to be unique, therefore threads with the same name cannot be compared.

Limitations:

* Gathering monitor details on some Sun VMs is slow when threads are suspended. It is possible to disable thread suspension by adding the Agent option threaddumpsuspendthreads=false. However, with thread suspension disabled, monitor details might not match the thread states or stack traces.
* CPU time is available only with JVM 5.0 and later.
* Acquiring owned monitors under .NET is available only with .NET 4.5.

Overview Tab:

* Use the Threads dashlet to create or open thread dumps and analyze the thread stacks.
* This view is divided into two areas. The upper area shows existing thread dumps for certain Agents. The bottom area shows information about the selected dump or the possibility to invoke a thread dump, if no dump is selected in the upper area.
* Use the Tasks Tab to modify or add tasks that can automatically create periodic thread dumps.

Thread Performance Metrics:

* Name: Name of the thread.
* Threads: Number of threads.
* Group: Name of the thread group the thread belongs to. This column may be empty because not all threads must belong to a group.
* CPU Time [ms]: Time, in milliseconds, that the thread used the CPU. (This information is only available for J2SE 5.0 and higher)
* State: Current state of the thread. There are six defined states:
* New: A that is created, but has not been started. There is no stack trace available for this thread.
* Runnable: Ready to execute or currently executing code.
* Blocked: A thread waiting to gain a monitor.
* Waiting: A thread that has yet to be notified, is sleeping, or is parked.
* Timed waiting: A thread that is waiting a fixed period of time.
* Terminated: A thread that has finished.
* Waiting On: Monitor (if any) that the thread is waiting on.
* Owned By: Owner of the monitor that this thread is waiting for (if any). A monitor may not have an owner.
* Owned Monitors: Number of monitors owned by this thread.
* Start Time: Time when the thread was started.
* Thread Id: JVM: Identity hashcode of the thread, for correlation. .NET: managed thread id

Stack Trace:

* #: Number of the stack trace element, i.e. the stack depth of the corresponding element.
* Class: Fully qualified name of the class, which contains the method that produced the stack trace element.
* Method: Name of the method.
* Line: Line number of the source file that is currently executing.
* Source File: Name of the source file for the executed method.

Owned Monitors:

* #: Number of the stack trace element, i.e. the stack depth of the corresponding element.
* Class: Fully qualified name of the class, which contains the method that produced the stack trace element.
* Method: Name of the method.
* Line: Line number of the source file that is currently executing.
* Source File: Name of the source file for the executed method.