
Command Reference: ProfileCommands()

Profile the commands that have executed, to evaluate performance

Version 11.09.02, 20163-03-23

The `ProfileCommands()` command summarizes run times and memory use for each command in the command list, and outputs the information to detail (row for each command) and summary (one row for each command name) tables. This command is useful for evaluating which commands are slow or use more memory in a command workflow, so that software and command file logic improvements can occur. The command is usually placed at the end of a command file. The following apply to command profiling:

- Currently profiling does not handle commands in `For()` command loops. Only the profile data for the last time the command is run will be saved.
- Because the command is processed at the time it is encountered in the command list, the command itself and any subsequent commands are not included in the analysis. This generally is not an issue because the command will be used near the end of a workflow or at a strategic location where previous commands need to be examined.
- Currently the memory statistics are rough because the heap size is determined at the start and end of each command's execution and the Java runtime environment may allocate heap memory in blocks. In the future profiling data may be expanded to the estimated memory footprint of each command.
- There is a slight performance and memory hit to collect profiling information. In the future processor property commands may be implemented to control how much profiling data are collected (specifically if memory for each command object is estimated).
- If a command file is causing out of memory exceptions, then placing a `ProfileCommands()` command at the end of the command file likely will not be helpful. Instead, use a subset of the full command list so the `ProfileCommands()` command will be executed. Then evaluate the performance of the commands and determine if the command list logic can be optimized. If performance issues appear to be in the software itself, contact the developers to evaluate the software code. Also consider using the `Free()` and `FreeTable()` commands to free resources, especially if the results do not need to be available to users via the user interface.
- The runtime percent for each command is calculated as a percentage of the total runtime (ignoring the `ProfileCommands()` command and subsequent commands).
- The heap memory percentage delta for each command is calculated using the heap memory at the end of the command execution (not the heap memory at the end of the full run). Consequently, the delta reflects the memory use up to that point in time.
- Command profiling currently only applies to run mode. Commands are executed in discovery mode when a command file is loaded. For example, a subset of time series data is retrieved so that time series identifiers can be created and passed to following commands, which allows choices to be populated in command editors. Profiling discover mode is not currently supported but should use a fraction of full runtime resources. For large command files (e.g., those generated by templates), it may be appropriate or necessary to load the commands without running discovery (see the `-nodiscovery` command line parameter and the **File...Open...Command File (no discovery)** menu item).
- Commands that generate many warning and failure messages will use more memory. Refer to the `NumLogRecords` column in the detail table to determine if this could be causing memory issues.

- The command currently does not allow sorting output tables by a column. This feature may be added in the future. Use the interactive table view to sort by column (this is how the tables were sorted for the figures below).

If loading or running commands are slow, the following actions might help:

- Use the `Free()` and `FreeTable()` commands to free resources. The command will still take up some resources because it has a place in the command list, but data resources used by the command will be freed.
- Review the profiling results to determine if certain commands are major resource users. Evaluate whether changes in the command logic can be implemented. Comment out blocks of commands (# commands will take fewer resources than `/* */` blocks because commands within the latter are still loaded rather than simple comment commands) and try to isolate problems. It may be necessary to run smaller subsets of commands, for example by splitting up lists of input time series.
- On Windows, use the Task Manager (run `taskmgr`) to review memory use by the `javaw.exe` program. If the memory use approaches the maximum, then the Java Runtime Environment likely will be spending time dealing with short memory and runtimes will increase until memory runs out. If necessary, change the `-Xmx` parameter in the `TSTool.l4j.ini` file located in the system folder under the software install. This parameter indicates the maximum heap memory that can be used by the software. For a typical 32-bit Windows computer with at least 4GB of memory, the `-Xmx` parameter may be set to as high as `1700mb`; however, a number that is too high may not be possible due to memory being used by other applications on the computer.

The following dialog is used to edit the command and illustrates the syntax of the command.

Edit ProfileCommands() Command

This command profiles commands and saves execution time and memory information in detail and summary tables. If the software is running out of memory before completing the commands, then use this command on a subset of commands to identify memory usage. The profile information for this command and following commands will be incomplete (because this command does not know about later commands). Commands within a `For()` loop are currently only profiled for the last executed iteration.

Summary table ID: Optional - unique identifier for the summary table.

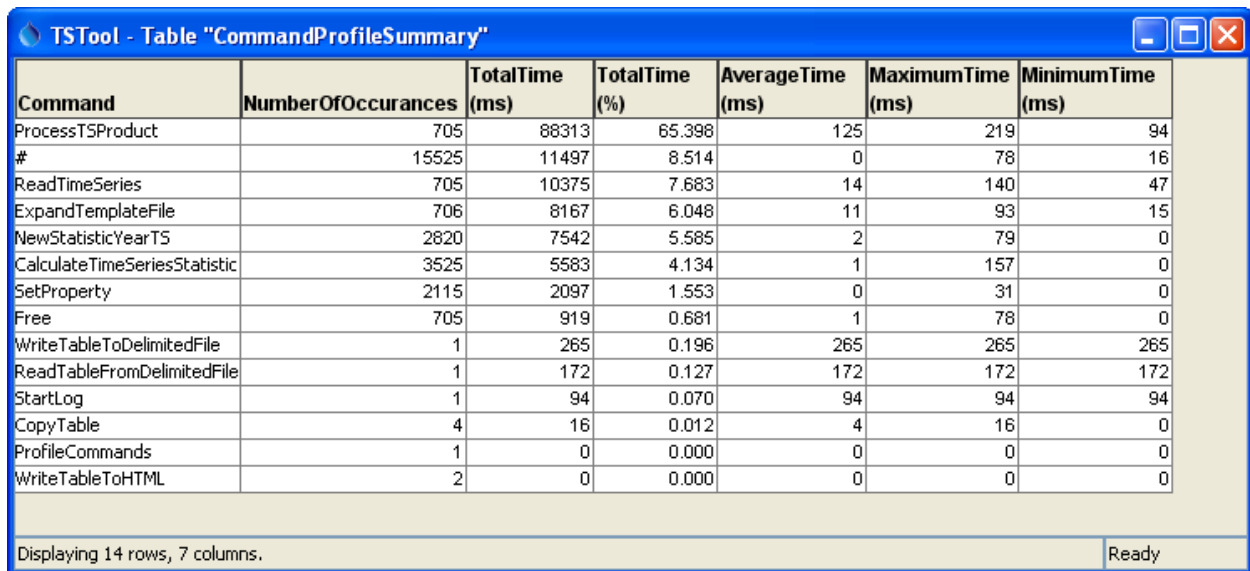
Detail table ID: Optional - unique identifier for the detail table.

Command:

ProfileCommands

ProfileCommands() Command Editor

The following figure illustrates the output summary table. Because command execution may be very fast, times are shown in milliseconds (1/1000th of a second). The table can be output to a file with other commands.



TSTool - Table "CommandProfileSummary"

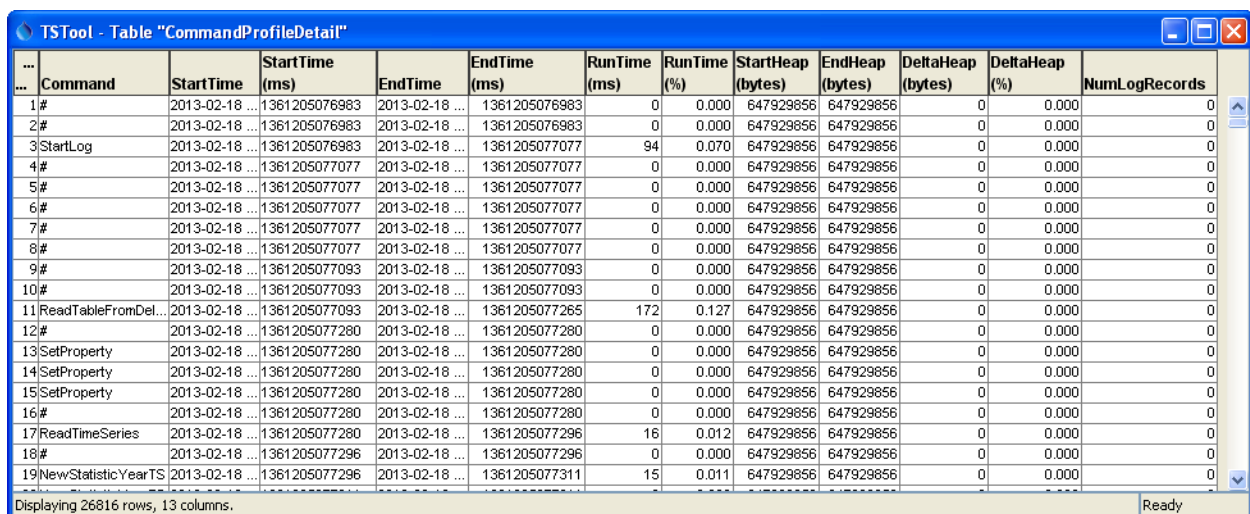
Command	NumberOfOccurrences	TotalTime (ms)	TotalTime (%)	AverageTime (ms)	MaximumTime (ms)	MinimumTime (ms)
ProcessTSProduct	705	88313	65.398	125	219	94
#	15525	11497	8.514	0	78	16
ReadTimeSeries	705	10375	7.683	14	140	47
ExpandTemplateFile	706	8167	6.048	11	93	15
NewStatisticYearTS	2820	7542	5.585	2	79	0
CalculateTimeSeriesStatistic	3525	5583	4.134	1	157	0
SetProperty	2115	2097	1.553	0	31	0
Free	705	919	0.681	1	78	0
WriteTableToDelimitedFile	1	265	0.196	265	265	265
ReadTableFromDelimitedFile	1	172	0.127	172	172	172
StartLog	1	94	0.070	94	94	94
CopyTable	4	16	0.012	4	16	0
ProfileCommands	1	0	0.000	0	0	0
WriteTableToHTML	2	0	0.000	0	0	0

Displaying 14 rows, 7 columns. Ready

ProfileCommands_Summary

ProfileCommands() Command Summary Output Table

The following figure illustrates the output detail table. Note that the heap memory is increased in blocks by the Java Runtime Environment so only large memory footprint commands trigger immediate heap memory increases.



TSTool - Table "CommandProfileDetail"

...	Command	StartTime	StartTime (ms)	EndTime	EndTime (ms)	RunTime (ms)	RunTime (%)	StartHeap (bytes)	EndHeap (bytes)	DeltaHeap (bytes)	DeltaHeap (%)	NumLogRecords
1	#	2013-02-18 ...	1361205076983	2013-02-18 ...	1361205076983	0	0.000	647929856	647929856	0	0.000	0
2	#	2013-02-18 ...	1361205076983	2013-02-18 ...	1361205076983	0	0.000	647929856	647929856	0	0.000	0
3	StartLog	2013-02-18 ...	1361205076983	2013-02-18 ...	1361205077077	94	0.070	647929856	647929856	0	0.000	0
4	#	2013-02-18 ...	1361205077077	2013-02-18 ...	1361205077077	0	0.000	647929856	647929856	0	0.000	0
5	#	2013-02-18 ...	1361205077077	2013-02-18 ...	1361205077077	0	0.000	647929856	647929856	0	0.000	0
6	#	2013-02-18 ...	1361205077077	2013-02-18 ...	1361205077077	0	0.000	647929856	647929856	0	0.000	0
7	#	2013-02-18 ...	1361205077077	2013-02-18 ...	1361205077077	0	0.000	647929856	647929856	0	0.000	0
8	#	2013-02-18 ...	1361205077077	2013-02-18 ...	1361205077077	0	0.000	647929856	647929856	0	0.000	0
9	#	2013-02-18 ...	1361205077093	2013-02-18 ...	1361205077093	0	0.000	647929856	647929856	0	0.000	0
10	#	2013-02-18 ...	1361205077093	2013-02-18 ...	1361205077093	0	0.000	647929856	647929856	0	0.000	0
11	ReadTableFromDel...	2013-02-18 ...	1361205077093	2013-02-18 ...	1361205077265	172	0.127	647929856	647929856	0	0.000	0
12	#	2013-02-18 ...	1361205077280	2013-02-18 ...	1361205077280	0	0.000	647929856	647929856	0	0.000	0
13	SetProperty	2013-02-18 ...	1361205077280	2013-02-18 ...	1361205077280	0	0.000	647929856	647929856	0	0.000	0
14	SetProperty	2013-02-18 ...	1361205077280	2013-02-18 ...	1361205077280	0	0.000	647929856	647929856	0	0.000	0
15	SetProperty	2013-02-18 ...	1361205077280	2013-02-18 ...	1361205077280	0	0.000	647929856	647929856	0	0.000	0
16	#	2013-02-18 ...	1361205077280	2013-02-18 ...	1361205077280	0	0.000	647929856	647929856	0	0.000	0
17	ReadTimeSeries	2013-02-18 ...	1361205077280	2013-02-18 ...	1361205077296	16	0.012	647929856	647929856	0	0.000	0
18	#	2013-02-18 ...	1361205077296	2013-02-18 ...	1361205077296	0	0.000	647929856	647929856	0	0.000	0
19	NewStatisticYearTS	2013-02-18 ...	1361205077296	2013-02-18 ...	1361205077311	15	0.011	647929856	647929856	0	0.000	0

Displaying 26816 rows, 13 columns. Ready

ProfileCommands_Detail

ProfileCommands() Command Detail Output Table

The command syntax is as follows:

```
ProfileCommand(Parameter=Value,...)
```

Command Parameters

Parameter	Description	Default
SummaryTableID	The identifier for the summary table. Can be specified using <code>\${Property}</code> .	Summary table will not be created.
DetailTableID	The identifier for the detail table. Can be specified using <code>\${Property}</code> .	Detail table will not be created.