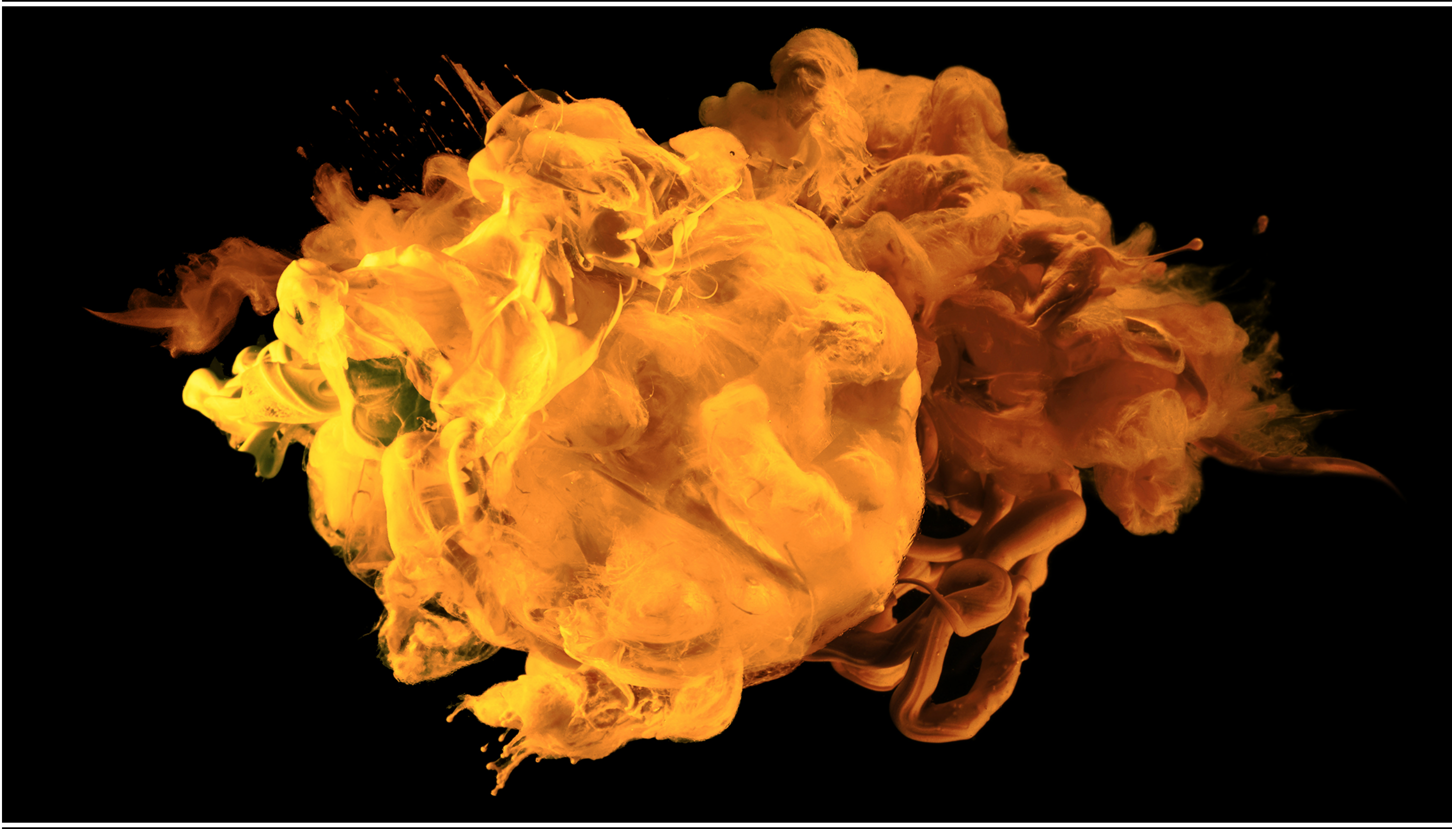
Application Tuning Guide

Dyalog version 18.0



DYALOG

The tool of thought for software solutions

Application Tuning Guide

Dyalog version 18.0
Document Revision: 20200115_180

Unless stated otherwise, all examples in this document assume that `⎕IO ⎕ML ← 1`

email: support@dyalog.com
https://www.dyalog.com

Contents

1 About This Document

This document describes the way in which the ⎕PROFILE system function and the associate]PROFILE user command can be used to obtain a performance profile of an application. It describes both the graphical and textual output that can be obtained following data collection and shows how this data can be analysed and potential inefficiencies identified..

1.1 Audience

It is assumed that the reader has a reasonable understanding of Dyalog.

For information on the resources available to help develop your Dyalog knowledge, see https://www.dyalog.com/introduction.htm.

1.2 Conventions

Unless explicitly stated otherwise, all examples in Dyalog documentation assume that ⎕IO and ⎕ML are both 1.

Various icons are used in this document to emphasise specific material.

General note icons, and the type of material that they are used to emphasise, include:

Hints, tips, best practice and recommendations from Dyalog Ltd.

Information note highlighting material of particular significance or relevance.

Legacy information pertaining to behaviour in earlier releases of Dyalog or to functionality that still exists but has been superseded and is no longer recommended.

Warnings about actions that can impact the behaviour of Dyalog or have unforeseen consequences.

Although the Dyalog programming language is identical on all platforms, differences do exist in the way some functionality is implemented and in the tools and interfaces that are available. A full list of the platforms on which Dyalog version 18.0 is supported is available at https://www.dyalog.com/dyalog/current-platforms.htm. Within this document, differences in behaviour between operating systems are identified with the following icons (representing macOS, Linux, UNIX and Microsoft Windows respectively):

2 Introduction

Application design includes assumptions about usage patterns and data volumes. Over time, these can evolve to the detriment of the application's performance. The most effective way to counter drops in performance caused by changes external to the application is to identify the hot spots (places in the application where a high proportion of CPU or Elapsed Time is consumed); these hot spots can then be tuned to improve the application's performance.

The `⎕PROFILE` system function and the `]Profile` user command facilitate the hot spot identification process; the `⎕PROFILE` system function gathers statistics from an application and the `]Profile` user command summarises, filters and reports on this data, simplifying the process of drilling down on the (frequently large amounts of) data returned by `⎕PROFILE`.

The `⎕MONITOR` system function, which was in use prior to the introduction of the `⎕PROFILE` system function, has been deprecated and Dyalog Ltd recommends rewriting tools to use the `⎕PROFILE` system function instead; `⎕PROFILE` provides high precision timing, calling tree analysis, and superior dfn and recursive code handling.

For more information on the `⎕PROFILE` system function, see the *Dyalog APL Language Reference Guide*. For more information on the `]Profile` user command, see *Appendix A*.

3 Data Collection

The `⎕PROFILE` system function can collect very large quantities of data. This means that, to profile a large application or to save a dataset as an XML file, the workspace size might need to be increased significantly.

Usage data is collected for all APL functions that are executed when the `⎕PROFILE` system function is in an active state.

For complete documentation of the `⎕PROFILE` system function, see the *Dyalog APL Language Reference Guide*.

3.1 Before Initiating the Collection of Data

To improve the accuracy of the data and minimise the impact of timer overhead (see *Section 3.5*):

- Switch off as much hardware as possible, including peripherals and network connections.
- Switch off as many other tasks and processes as possible, including anti-virus software, firewalls, internet services and background tasks.
- Raise the priority on the Dyalog APL task to higher than normal.

  On Microsoft Windows, avoid giving it the highest priority.

Data collected by the `⎕PROFILE` system function is cumulative whenever the `⎕PROFILE` system function is in an active state (but does not persist between Sessions); to discard any previously-collected data, enter `⎕PROFILE 'clear'`.

3.2 Initiating the Collection of Data

Data collection is initiated by entering:

```
      ⎕PROFILE 'start'
```

This puts ⎕PROFILE into an active state.

⎕PROFILE supports profiling using either CPU or elapsed time. CPU time is usually of more interest in profiling application performance and, by default, ⎕PROFILE will register CPU usage data using the best available counter.

3.3 Collecting Data

Whether data is being collected or not can be verified by entering:

```
      ⎕PROFILE 'state'
```

This returns a 4 element vector, in which:

- [1] is a character vector indicating the state of ⎕PROFILE.
  Can be either active or inactive (must be active for data collection).
- [2] is a character vector indicating the timer being used.
  Can be CPU, elapsed, coverage or none.
- [3] is the call time bias in milliseconds, that is the amount of time that is consumed when the system takes a time measurement.
- [4] is the timer granularity in milliseconds, that is, the resolution of the timer being used. On most platforms this will be zero, indicating that the granularity is smaller than the cost and cannot be estimated.

During data collection, the following data is recorded for each function and for each individual line in a function:

- Calls – the number of times the function or line was executed.
- Exclusive Time – milliseconds spent executing the function or line, excluding time spent in functions that were called by the function or line.
- Inclusive Time – milliseconds spent executing the function or line, including time spent in functions that were called by the function or line.

The times collected by the ⎕PROFILE system function include the time spent calling the timer function. This means that lines that are called a large number of times can appear to consume more resource than they actually do. For more accurate profiling measurements, adjustments should be made for the timer call time bias. To do this, the application should be run for a sufficiently long period to collect enough data to overcome the timer granularity – a reasonable rule of thumb is to make sure the application runs for at least 4000×4⊃⎕PROFILE 'state' milliseconds.

The profiling data that is collected is stored outside the workspace and does not impact workspace availability.

⎕PROFILE can collect data for functions that are dynamically paged in and out of the workspace.

Results can be confusing if several different functions with the same name are used at different times during execution – these are treated as the same function by ⎕PROFILE.

3.4 Stopping the Collection of Data

Data collection is stopped by entering:

```
      ⎕PROFILE 'stop'
```

This puts ⎕PROFILE into an inactive state.

3.5 Timer Overhead

As with all system timers, a cost is associated with the collection of timing data using the ⎕PROFILE system function. In optimised applications the overhead can be significant; although it is unlikely to impact the identification of hot spots, it can distort results.

By default, reports produced with the `]Profile` user command automatically adjust for timer bias, using the recorded bias – this can be disabled for a report by including the `-bias=0` modifier and modifier value.

The cost of querying a timer can vary significantly with system load, and repeatable timings are only possible if there is very little activity on the system; the variability is due to the timer calling the operating system kernel, which is servicing all processes on the machine. Dyalog Ltd. recommends increasing the priority of the application being profiled as this can reduce the variability (but will not eliminate it completely).

The timer cost and granularity are estimated the first time that ⎕PROFILE is run in an APL session. A new calibration can be requested by calling `⎕PROFILE 'calibrate'`.

4 Data Reporting

Once data has been collected, summarised data for each function and line within that function can be retrieved using `⎕PROFILE 'data'`. Alternatively, data can be broken down by calling trees so that it is summarised separately for every different path that led to the function being executed – this is done using `⎕PROFILE 'tree'`. In both these cases, a very large quantity of data can be returned.

The `]Profile` user command is a reporting tool that acts on the profiling data collected by the `⎕PROFILE` system function, applying filters and limiting the output to a specified subset of the total collected. This results in a reduced data quantity that is tailored to display only what is required. For details of the syntax of the `]Profile` user command, see *Appendix A*.

The examples in this document assume that data has been collected as follows:

```
      )LOAD sharpplot
C:\...\ ws\sharpplot.dws saved Mon May  8 09:57:02 2017

      ⎕PROFILE 'start'

      #.Samples.Sample 'Sample.svg'
mySharpPlot  Sample.svg

      ⎕PROFILE 'stop'
```

4.1 Textual Reports

After collecting the data, the consumption by function can be found using:

```
      ]Profile summary -expr="#.Samples.Sample 'Sample.svg'"
-first=10
 Total time: 58.2 msec

 Element                               msec       %  Calls
 #.Samples.Sample                      58.2   100.0      1
 #.SharpPlot.DrawLineGraph             25.6    44.0      1
 #.SharpPlot.Plot                      18.4    31.6      1
 #.SharpPlot.DrawBarChart               7.7    13.3      1
 #.SharpPlot.CH∆PLOT                    3.2     5.4      1
 #.SharpPlot.DrawPieChart               2.2     3.8      1
 #.SharpPlot.CH∆PIE                     1.9     3.3      1
 #.SharpPlot.ConstructorDefault         1.3     2.2      1
 #.SharpPlot.SharpPlot                  1.3     2.2      1
 #.psb.Constructor                      1.2     2.1     12
```

In this expression the `-expr` modifier allows the specification of an APL statement to run, and is equivalent to executing:

```
      ⎕PROFILE 'clear'
      ⎕PROFILE 'start'
      #.Samples.Sample 'Sample.svg'
      ⎕PROFILE 'stop'
```

The inclusion of `-first=10` limits the output to the top 10 functions in terms of CPU consumption. To see the top 5 lines of code instead:

```
      ]Profile summary -lines -first=5
 Total time: 58.2 msec

 Element                              msec      %  Calls
 #.Samples.Sample[33]                 25.9   44.5      1
 #.Samples.Sample[42]                 25.6   44.0      1
 #.SharpPlot.DrawLineGraph[43]        25.6   43.9      1
 #.SharpPlot.Plot[174]                17.7   30.5      1
 #.Samples.Sample[60]                  2.2    3.8      1
```

Finally, the call analysis report for the `Constructor` function can be displayed:

```
      ]Profile calls -fn=#.psb.Constructor -first=5
 Total time: 58.2 msec; Selected time: 1248.0 msec

 Element                    msec     %  Calls
 #.SharpPlot.CH∆PLOT         0.4   0.8      3
 #.SharpPlot.SharpPlot       0.3   0.6      4
 #.SharpPlot.CH∆BAR          0.2   0.3      2
 #.SharpPlot.CH∆AXES         0.1   0.2      1
 #.SharpPlot.CH∆XTIC         0.1   0.2      1
```

4.2 Graphical Reports

The Dashboard detailed in this section is only available on the Microsoft Windows operating system. For information on using the Dashboard, see *Appendix B*.

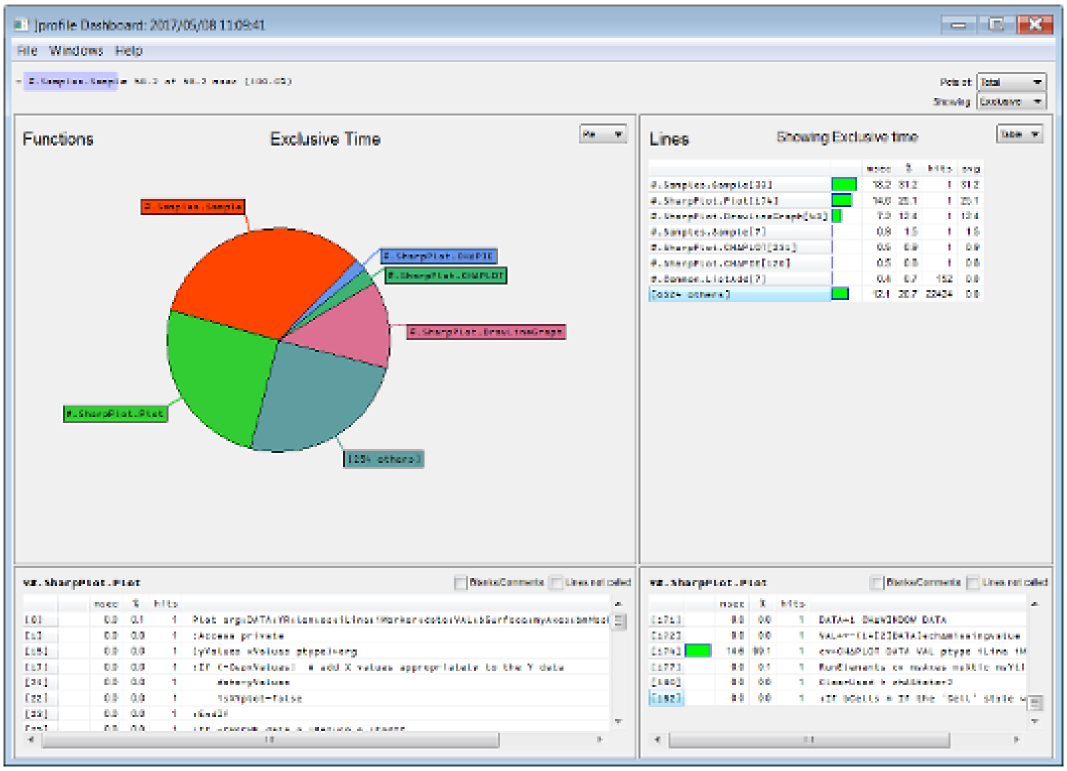
A graphical version of the textual reports can be viewed using the Dashboard. This provides an overview of the resource consumption of an application that can be drilled down into in pursuit of tuning opportunities.

To open the Dashboard on the dataset, call the `]Profile` user command without specifying any report type, that is:

```
      ]Profile
```

The Dashboard will open and display an overview of the data currently stored by `⎕PROFILE` (`⎕PROFILE` must be stopped/inactive), as shown in *Figure 4-1*.

If the dataset is very large then it can take a few seconds to open the Dashboard.



		msec	%	hits	avg
#.Samples.Sample[33]	🟩	18.2	31.2	1	31.2
#.SharpPlot.Plot[174]	🟩	14.6	25.1	1	25.1
#.SharpPlot.DrawLineGraph[43]	🟩	7.3	12.4	1	12.4
#.Samples.Sample[7]		0.8	1.5	1	1.5
#.SharpPlot.CHAPLOT[331]		0.5	0.8	1	0.8
#.SharpPlot.CHAPTER[126]		0.5	0.8	1	0.8
#.Common.ListAdd[7]		0.4	0.7	152	0.0
[3324 others]	🟩	12.1	20.7	22434	0.0

```
#.SharpPlot.Plot
            msec    %   hits
[0]          0.0  0.1     1  Plot arg;DATA;YR;Lenses;Line;Marker;Color;VAL;Surface;myAxes;mMod
[1]          0.0  0.0     1  :Access private
[15]         0.0  0.0     1  (yValues xValues ptype)←arg
[17]         0.0  0.0     1  :If (~0=xValues)  ⍝ add X values appropriately to the Y data
[21]         0.0  0.0     1      data←yValues
[22]         0.0  0.0     1      isXYplot←false
[23]         0.0  0.0     1  :EndIf
```

```
#.SharpPlot.Plot
            msec    %   hits
[171]        0.0  0.0     1  DATA←1 DRAWINDOW DATA
[172]        0.0  0.0     1  VAL←⍬⍴[1]←[2]DATA]←chartlineinpvalue
[174]  🟩   14.6 99.1     1  c←←CHAPLOT DATA VAL ptype ⍝Line ⍝M
[177]        0.0  0.1     1  RunElements c← mdues mdRlc mdYLl
[180]        0.0  0.0     1  ClearUsed ⍝ endWater3
[182]        0.0  0.0     1  :If NCells ⍝ If the 'Cell' state w
```

Figure 4-1: *Dashboard showing default configuration*

To open the Dashboard on a dataset that is not currently stored by ⎕PROFILE, the modifiers `-expr` and/or `-infile` can be included:

- `-expr` runs the specified expression and then opens the Dashboard on the resultant dataset, for example, `]Profile -expr="#.Samples.Sample 'Sample.svg'"`. Doing this destroys any existing ⎕PROFILE data and replaces it with data for the specified expression.
- `-infile` opens the Dashboard on the dataset contained in the specified **.xml** file , for example, `]Profile -infile=c:\temp\one.xml`. Doing this does not destroy any existing ⎕PROFILE data.

5 Data Storage

The `]Profile` user command can direct output to a file instead of displaying it in the Session. To do this, the `-outfile` modifier must be included; its modifier value must be the name of file in which to write the report data. By default, data is stored in an XML format, but this can be changed with the `-format` modifier:

- To save data in XML format, include `-format=xml` in the call to the `]Profile` user command (see *Section 5.1*). This is the default, so does not have to be explicitly stated.
- To save data in CSV format, include `-format=csv` in the call to the `]Profile` user command (see *Section 5.2*).
- To save data in text format, include `-format=txt` in the call to the `]Profile` user command (see *Section 5.3*).

5.1 XML Format

Using the XML format generates very large files. However, `tree` reports in XML format can be used as input to the `]Profile` user command (using the `-infile` modifier) and are the only way to store a complete data set that can be reused for reporting at a later time. For example, entering the following command:

```
]Profile tree -outfile=c:\temp\one.xml
```

saves the tree report in a file called **one.xml** in the **c:\temp** directory. This file can be opened later (in the same or a different Session) by entering:

```
]Profile summary -infile=c:\temp\one.xml
```

In addition, as user commands can be executed under program control, an application can record its own usage data. For example:

```
⎕SE.UCMD 'profile tree -outfile=c:\temp\one.xml'
```

The XML format produced by the `]Profile` user command comprises an outer `<ProfileData>` element; this contains a `<ProfileSettings>` element followed by a number of `<ProfileEntry>` elements, one for each row of output data.

The `<ProfileSettings>` element contains the version number of the `]Profile` user command that produced the file, the report title, information about timer cost and other information, including the total registered time for the report.

Each `<ProfileEntry>` element contains an element for each output column, depending on the command and switches, selected from the set listed in *Table 5-1*.

***Table 5-1:** Elements that can be contained within the `ProfileEntry` element*

Element	Description
`Depth`	Tree depth
`Element`	Function name
`Line`	Line number – empty for a function summary entry
`Calls`	Number of times the function or line was called
`IncusiveTime` `ExclusiveTime`	Time consumed inclusive/exclusive of time consumed in any sub-functions called (in ms)
`AvgTime`	Average time per call (in ms)

5.1.1 Example XML Files

This section contains a few examples of output files created using `-format=xml` (all files are encoded as UTF-8).

```
      ]Profile tree -outfile=tree.xml
Data written to: tree.xml
```

Content of the **tree.xml** file:

```
<?xml version="1.0"?>
<ProfileData>
    <ProfileSettings>
        <Version>1.37</Version>
        <Title>2017/05/08 11:22:19</Title>
        <TimerBias>0.00007458669726290168</TimerBias>
        <Command>tree</Command>
        <TotalTime>173.41</TotalTime>
        <SelectedTime>173.41</SelectedTime>
    </ProfileSettings>
    <ProfileEntry>
        <Depth>0</Depth>
        <Element>#.Samples.Sample</Element>
        <Line>¯1</Line>
        <Calls>1</Calls>
        <ExclusiveTime>19.362</ExclusiveTime>
        <InclusiveTime>61.828</InclusiveTime>
    </ProfileEntry>
    ...
    ...many more occurrences of <ProfileEntry>...
    ...
</ProfileData>
```

```
      ]Profile summary -outfile=summary.xml
Data written to: summary.xml
```

Content of the **summary.xml** file:

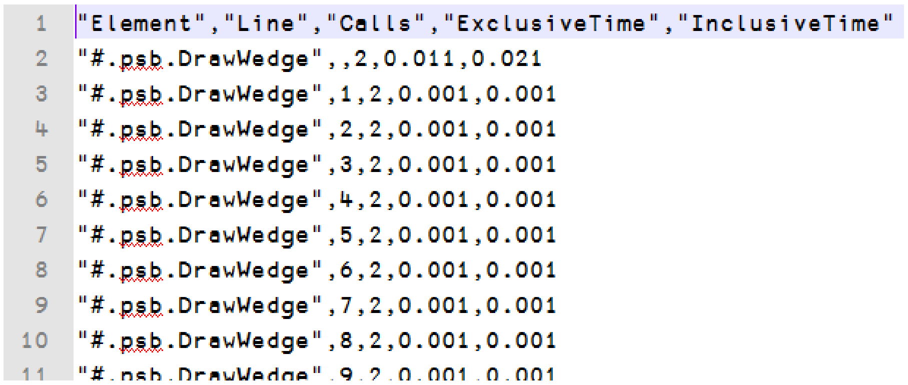
```
<?xml version="1.0"?>
<ProfileData>
    <ProfileSettings>
        <Version>1.37</Version>
        <Title>2017/05/08 11:31:00</Title>
        <TimerBias>0.00007458669726290168</TimerBias>
        <Command>summary</Command>
        <TotalTime>58.2</TotalTime>
        <SelectedTime>58.2</SelectedTime>
    </ProfileSettings>
    <ProfileEntry>
        <Element>#.Samples.Sample</Element>
        <Line/>
        <InclusiveTime>58.2</InclusiveTime>
        <PctOfTot>100</PctOfTot>
        <Calls>1</Calls>
    </ProfileEntry>
    ...
    ...many more occurrences of <ProfileEntry>...
    ...
</ProfileData>
```

5.2 CSV Format

Files saved in CSV format can be used by many external tools. For example:

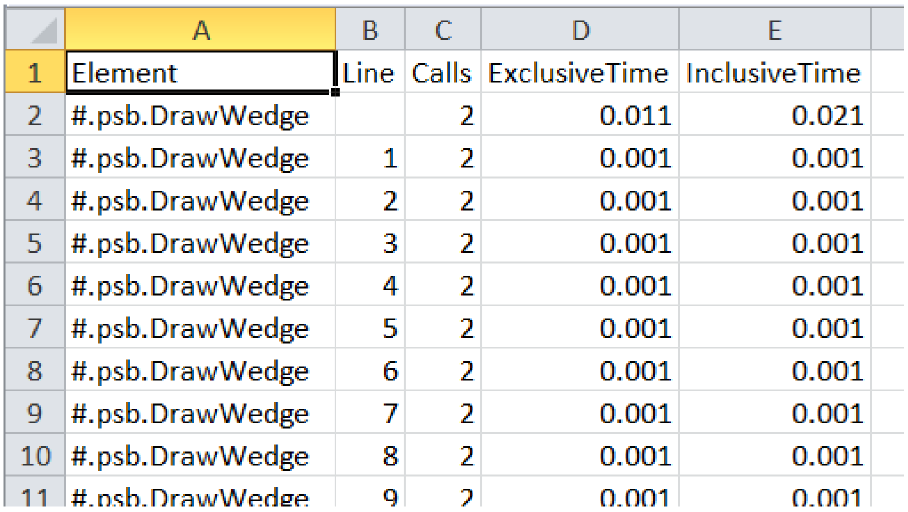
```
      ]Profile data -outfile=c:\temp\data.csv -format=csv
-separators='.,'
```

creates a CSV file using a period as the decimal separator and a comma as the field separator. This file can be viewed either by opening it in a text editor or in a spreadsheet (as shown in *Figure 5-1* and *Figure 5-2* respectively).



```
"Element","Line","Calls","ExclusiveTime","InclusiveTime"
"#.psb.DrawWedge",,2,0.011,0.021
"#.psb.DrawWedge",1,2,0.001,0.001
"#.psb.DrawWedge",2,2,0.001,0.001
"#.psb.DrawWedge",3,2,0.001,0.001
"#.psb.DrawWedge",4,2,0.001,0.001
"#.psb.DrawWedge",5,2,0.001,0.001
"#.psb.DrawWedge",6,2,0.001,0.001
"#.psb.DrawWedge",7,2,0.001,0.001
"#.psb.DrawWedge",8,2,0.001,0.001
"#.psb.DrawWedge",9,2,0.001,0.001
```

Figure 5-1: *Viewing the saved data CSV file as a text file*



	A	B	C	D	E
1	Element	Line	Calls	ExclusiveTime	InclusiveTime
2	#.psb.DrawWedge		2	0.011	0.021
3	#.psb.DrawWedge	1	2	0.001	0.001
4	#.psb.DrawWedge	2	2	0.001	0.001
5	#.psb.DrawWedge	3	2	0.001	0.001
6	#.psb.DrawWedge	4	2	0.001	0.001
7	#.psb.DrawWedge	5	2	0.001	0.001
8	#.psb.DrawWedge	6	2	0.001	0.001
9	#.psb.DrawWedge	7	2	0.001	0.001
10	#.psb.DrawWedge	8	2	0.001	0.001
11	#.psb.DrawWedge	9	2	0.001	0.001

Figure 5-2: *Viewing the saved data CSV file as a Microsoft Excel spreadsheet*

Figure 5-2 shows the **data.csv** file opened in Microsoft Excel; to do this, enter the following in a Session:

```
      'XL' ⎕WC 'OLEClient' 'Excel.Application'
      XL.Visible←1
      XL.Workbooks.Open⊂'c:\temp\data.csv'
```

5.2.1 Example CSV Files

This section contains a few examples of output files created using `-format=csv` (all files are encoded as UTF-8). The first row of each file contains column names, selected from the same list as the element names that can appear in XML files (see *Table 5-1*).

```
      ]Profile tree -outfile=data1.csv -format=csv
Data written to: data1.csv
```

Content of the **data1.csv** file:

```
"Depth","Element","Line","Calls","ExclusiveTime","InclusiveTime"
0,"#.Samples.Sample",¯1,1,19.362,61.828
1,"#.Samples.Sample",1,1,0.002,0.002
1,"#.Samples.Sample",2,1,0,0
1,"#.Samples.Sample",3,1,0,0
1,"#.Samples.Sample",4,1,0,0
1,"#.Samples.Sample",5,1,0.005,0.005
1,"#.Samples.Sample",6,1,0,0
1,"#.Samples.Sample",7,1,0.874,2.388
2,"#.SharpPlot.ConstructorDefault",¯1,1,0.007,1.513
3,"#.SharpPlot.ConstructorDefault",1,1,0,0
3,"#.SharpPlot.ConstructorDefault",2,1,0.001,0.001
3,"#.SharpPlot.ConstructorDefault",3,1,0.004,1.51
4,"#.SharpPlot.SharpPlot",¯1,1,0.219,1.506
...etc...
```

```
      ]Profile data -outfile=data2.csv -format=csv
-separators=",;"
Data written to: data2.csv
```

Content of the **data2.csv** file:

```
"Element";"Line";"Calls";"ExclusiveTime";"InclusiveTime"
"#.psb.DrawWedge";;2;0,011;0,021
"#.psb.DrawWedge";1;2;0,001;0,001
"#.psb.DrawWedge";2;2;0,001;0,001
"#.psb.DrawWedge";3;2;0,001;0,001
"#.psb.DrawWedge";4;2;0,001;0,001
"#.psb.DrawWedge";5;2;0,001;0,001
"#.psb.DrawWedge";6;2;0,001;0,001
"#.psb.DrawWedge";7;2;0,001;0,001
```

```
"#.psb.DrawWedge";8;2;0,001;0,001
"#.psb.DrawWedge";9;2;0,001;0,001
...etc...
```

```
      ]Profile summary -first=5 -outfile=data3.csv -format=csv
Data written to: data3.csv
```

Content of the **data3.csv** file:

```
"Element","Line","Time","PctOfTot","Calls"
"#.Samples.Sample",,58.2,100,1
"#.SharpPlot.DrawLineGraph",,25.626,44.03092784,1
"#.SharpPlot.Plot",,18.373,31.56872852,1
"#.SharpPlot.DrawBarChart",,7.726,13.27491409,1
"#.SharpPlot.CH∆PLOT",,3.153,5.417525773,1
```

5.3 Text Format

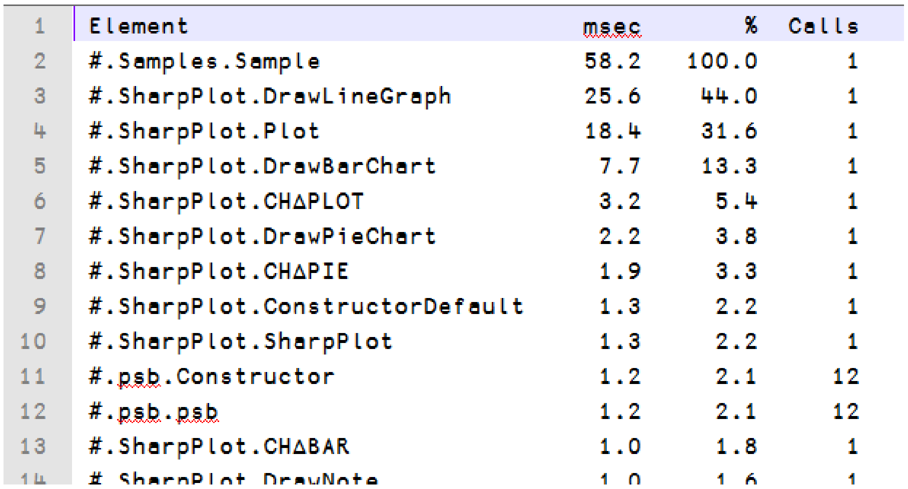
If the text format is used, the output is written to the file as it would have been displayed in the Session.

5.3.1 Example Text Files

```
      ]Profile summary
 Total time: 58.2 msec

 Element                              msec      %  Calls
 #.Samples.Sample                     58.2  100.0      1
 #.SharpPlot.DrawLineGraph            25.6   44.0      1
 #.SharpPlot.Plot                     18.4   31.6      1
 #.SharpPlot.DrawBarChart              7.7   13.3      1
 #.SharpPlot.CH∆PLOT                   3.2    5.4      1
 #.SharpPlot.DrawPieChart              2.2    3.8      1
 #.SharpPlot.CH∆PIE                    1.9    3.3      1
 #.SharpPlot.ConstructorDefault        1.3    2.2      1
 #.SharpPlot.SharpPlot                 1.3    2.2      1
 #.psb.Constructor                     1.2    2.1     12
 #.psb.psb                             1.2    2.1     12
 #.SharpPlot.CH∆BAR                    1.0    1.8      1
 #.SharpPlot.DrawNote                  1.0    1.6      1
 #.SharpPlot.CH∆XLAB                   0.9    1.6      1
 #.SharpPlot.RunElements               0.9    1.6      2
 #.SharpPlot.CH∆NOTE                   0.9    1.6      1
 #.psb.MeasureEach                     0.9    1.5      8
 #.SharpPlot.CH∆METRIC                 0.9    1.5      7
 #.Common.ListAdd                      0.8    1.4    152
 #.SharpPlot.CH∆HEAD                   0.8    1.3      2

      ]Profile summary -outfile=data.txt -format=txt
Data written to: data.txt
```



```
Element                              msec      %  Calls
#.Samples.Sample                     58.2  100.0      1
#.SharpPlot.DrawLineGraph            25.6   44.0      1
#.SharpPlot.Plot                     18.4   31.6      1
#.SharpPlot.DrawBarChart              7.7   13.3      1
#.SharpPlot.CH∆PLOT                   3.2    5.4      1
#.SharpPlot.DrawPieChart              2.2    3.8      1
#.SharpPlot.CH∆PIE                    1.9    3.3      1
#.SharpPlot.ConstructorDefault        1.3    2.2      1
#.SharpPlot.SharpPlot                 1.3    2.2      1
#.psb.Constructor                     1.2    2.1     12
#.psb.psb                             1.2    2.1     12
#.SharpPlot.CH∆BAR                    1.0    1.8      1
#.SharpPlot.DrawNote                  1.0    1.6      1
```

Figure 5-3: *Viewing the saved data txt file as a text file*

A Syntax of the]Profile User Command

The `]Profile` user command is always followed by a report type; modifiers can be included to customise the output.

Syntax:
```
]Profile [reporttype][-avg][-code][-lines]|[-outfile{=name}]
[-format{=xml|csv|txt}][-cumpct][-exclusive][-first{=n}|-pct{=n}]
[-fn{=name}][-infile{=name}][-separators{="decimalsep phrasesep"}]
[-bias{=t}][-decimal{=n}][-expr{=expression}][-title{=name}]
```

A.1 Report Types

The six possible report types are detailed in *Table A-1*. If no report type is specified then a default report type is assumed; this is dashboard on the Microsoft Windows operating system and summary on the AIX, Linux and Mac OS operating systems.

Table A-1: *Report types that can be generated using the `]Profile` user command*

Report Type	Description
`calls`	Shows how the consumption of a named function (the `-fn` modifier is required) is broken down by calling function. The `summary` and `calls` report types are the most frequently used reporting tools.
`dashboard`	The Dashboard is only available on the Microsoft Windows operating system. Opens the Dashboard, a graphical overview of the profiling data collected by the `⎕PROFILE` system function. For more information on the Dashboard, see *Appendix B*. This is the default report type on the Microsoft Windows operating system.

Table A-1: *Report types that can be generated using the]Profile user command (continued)*

Report Type	Description
`data`	Writes the raw data produced by `⎕PROFILE 'data'` to a file for use with tools other than `]Profile`, for example, Microsoft Excel.
`state`	Displays the current profiling state of `⎕PROFILE` (see *Section 3.3*).
`summary`	Reports the number of calls, total consumption and consumption as a percentage of overall consumption. The `summary` and `calls` report types are the most frequently used reporting tools. This is the default report type on the AIX, Linux and macOS operating systems.
`tree`	Writes the raw data produced by `⎕PROFILE 'tree'` to a file for later use. Intended as a tool for storing data using the `-outfile=<name>` modifier, for subsequent reporting using the `-infile=<name>` modifier.

A.2 Modifiers

The report types can be qualified using modifiers. These can, for example, filter the data that is displayed, add optional output columns, read input from a previously saved file or store the results of a command in a file.

Each of the report types can have different combinations of modifiers applied. The `state` report type does not take any modifiers; the valid modifiers for each of the other report types are shown in *Table A-2*.

Table A-2: *Report types and the modifiers that can be applied to them*

Modifier	Report Types				
	`calls`	`dashboard`	`data`	`summary`	`tree`
`-avg`	y		y	y	
`-bias`	y	y	y	y	y
`-code`	y			y	

***Table A-2:** Report types and the modifiers that can be applied to them (continued)*

Modifier	Report Types				
	`calls`	`dashboard`	`data`	`summary`	`tree`
`-cumpct`	y		y	y	
`-decimal`	y		y	y	y
`-exclusive`	y		y	y	
`-expr`	y	y	y	y	y
`-first`	y		y	y	
`-fn`	y	y	y	y	
`-format`	y		y	y	y
`-infile`	y	y	y	y	y
`-lines`	y		y	y	y
`-outfile`	y		y	y	y
`-pct`	y		y	y	
`-separators`	y*		y*	y*	y*
`-title**`	y**	y**	y**	y**	y**

* can only be used when `-format=csv` is included
** only relevant when `-format=xml` or `-format=txt`

Table A-3, *Table A-4* and *Table A-5* describe these modifiers.

***Table A-3:** Modifiers for data selection*

Modifier	Description
`-avg`	Includes the average CPU consumption (in ms) per execution of each function call (or line if the `-lines` modifier is specified).

Table A-3: *Modifiers for data selection (continued)*

Modifier	Description
`-code`	Includes the source code for the line being executed (including the `-code` modifier forces the `-lines` modifier). Cannot be used with the `-outfile` modifier.
`-cumpct`	Displays the cumulative percentage of overall CPU consumption that each function call (or line if the `-lines` modifier is specified) and each function call above it was responsible for. This is usually only useful if the `-exclusive` modifier is also set.
`-exclusive`	Displays the CPU consumption of each function call (or line if the `-lines` modifier is specified) excluding consumption due to called functions.
`-first=`*n*	After sorting into descending order of CPU consumption, displays only the first *n* function calls (or lines if the `-lines` modifier is specified). This is usually only useful if the `-exclusive` modifier is also set. Cannot be used with the `-pct` modifier.
`-fn=`*name*	Mandatory for a `calls` report type, when it specifies the function that the calls analysis report is for. Optional for other report types, when output is filtered to only include data for the specified function and other functions that it calls.
`-lines`	Displays a breakdown of consumption by individual line rather than a total for each function (the default). Assumed when the `-code` modifier is specified.
`-pct=`*n*	After sorting into descending order of CPU consumption, displays only those function calls (or lines if the `-lines` modifier is specified) for which the cumulative percentage of overall CPU consumption is less than or equal to *n*. This is usually only useful if the `-exclusive` modifier is also set. Cannot be used with the `-first` modifier.

Table A-4: *Modifiers for redirecting output to a file rather than display it on the screen*

Modifier	Description
`-format=`*n*	Selects the file format to use when saving a file using the `-outfile` modifier. Possible values are: • `xml` – writes data to an XML file • `csv` – writes data to a CSV file • `txt` – writes data to a text file (and retains the display format) The default is `xml`.
`-infile=`*n*	Opens the Dashboard on the dataset contained in the specified **.xml** file. Doing this does not destroy any existing `⎕PROFILE` data.
`-separators=`*nn*	For use with `-format=csv`. Specifies the decimal and comma separators to use. The default is `.,`
`-title=`*n*	For use with `-format=xml` or `-format=txt`. Specifies the string that is used as a title caption in the Dashboard and XML reports. Especially useful when running the same expression multiple times as different captions can differentiate between different sets of results. If the `-title` modifier is not specified, then the caption defaults to the string specified by the `-expr` modifier. If neither the `-title` nor the `-expr` modifiers are specified, then the caption defaults to **]profile Dashboard: <date> <time>**
`-outfile=`*n*	Redirects the output from the Session to the specified full path and filename (the full path must already exist). Cannot be used with the `-code` modifier.

Table A-5: *Other modifiers*

Modifier	Description
`-bias=`*t*	Overrides the function call overhead estimated by `⎕PROFILE` during the current session (or read from an `infile`), and uses *t* instead. Use `-bias=0` to ignore bias, or a fixed value if you want to make sure that you use the same bias for data collected at different times. Depending on environment, *t* is likely to be in the range 0.00001-0.001 (in ms).

Table A-5: *Other modifiers (continued)*

Modifier	Description
`-decimal=`*n*	Specifies the number of decimal places to display for non-integer numbers. The default is 1.
`-expr=`*n*	Executes the expression specified as the modifier value and replaces any existing `⎕PROFILE` data with that for the specified expression.

A.3 Examples

The examples in this section are intended to show at least one use of every modifier.

```
      )LOAD sharpplot
C:\...\ ws\sharpplot.dws saved Mon May  8 09:57:02 2017

      ⎕PROFILE 'start'

      #.Samples.Sample 'Sample.svg'
mySharpPlot  Sample.svg

      ⎕PROFILE 'stop'
```

To see which 5 functions consumed the most CPU time:

```
      ]Profile summary -expr="#.Samples.Sample 'Sample.svg'"
-first=5
 Total time: 56.1 msec

 Element                      msec       %  Calls
 #.Samples.Sample             56.1   100.0      1
 #.SharpPlot.DrawLineGraph    24.9    44.4      1
 #.SharpPlot.Plot             18.0    32.1      1
 #.SharpPlot.DrawBarChart      7.8    14.0      1
 #.SharpPlot.CH∆PLOT           3.2     5.6      1
```

Show the five biggest CPU consumers, excluding CPU time spent in sub-functions. Display decimal numbers to 3 decimal places, include a cumulative percentage and only include functions up to 65% of the cumulative CPU:

```
      ]Profile summary -exclusive -decimal=3 -cumpct -pct=65
 Total time: 58.2 msec

 Element                msec        %  Calls   %(cum)
 #.Samples.Sample     19.351   33.249      1   33.249
 #.SharpPlot.Plot     14.719   25.290      1   58.540
```

To see the average CPU consumption per call without adjusting for timer bias:

```
      ]Profile summary -exclusive -decimal=3 -avg -bias=0 -first=3
 Total time: 61.8 msec

 Element                         msec        %  Calls      Avg
 #.Samples.Sample              19.362   31.316      1   19.362
 #.SharpPlot.Plot              14.733   23.829      1   14.733
 #.SharpPlot.DrawLineGraph      7.254   11.733      1    7.254
```

The second set of numbers are higher than the first – the total time is 3.6 ms higher when the timer bias adjustment is not made and the function with the highest consumption, `#.Samples.Sample`, is reported as having consumed 0.011 ms more. The raw data recorded for a function can be displayed (without bias adjustment) by the data report type; in this case the function with the highest consumption is the one of interest:

```
      ]Profile data -fn=#.Samples.Sample
 Total time: 389.7 msec; Selected time: 104475.0 msec

 Element                 Calls  msec(exc)  msec(inc)
 #.Samples.Sample            1       23.0      104.5
 #.Samples.Sample[1]         1        0.0        0.0
 #.Samples.Sample[2]         1        0.0        0.0
 #.Samples.Sample[3]         1        0.0        0.0
 #.Samples.Sample[4]         1        0.0        0.0
 #.Samples.Sample[5]         1        0.0        0.0
 #.Samples.Sample[6]         1        0.0        0.0
 #.Samples.Sample[7]         1        1.4        2.3
 #.Samples.Sample[8]         1        0.0        0.0
 ...etc...
```

For a summary or calls report, the `-code` modifier can be used to include source code in a report:

```
      ]Profile summary -code -lines -first=5
 Total time: 56.1 msec

 Element                          msec      %  Calls  Code
 #.Samples.Sample[33]             25.5   45.4      1
sp.DrawBarChart⊂data1
 #.Samples.Sample[42]             24.9   44.4      1
sp.DrawLineGraph⊂data2
 #.SharpPlot.DrawLineGraph[43]    24.9   44.3      1  Plot yValues
xValues'linegraph'
 #.SharpPlot.Plot[174]            17.3   30.9      1  cv←CH∆PLOT
DATA VAL ptype iLine iMarker(bFramed∨bCropped)
 #.Samples.Sample[7]               2.0    3.5      1  sp←⎕NEW
Causeway.SharpPlot
```

The `-outfile` modifier allows output to be directed to a file instead of displaying it in the Session. By default, the format of the data in the file is XML, but this can be changed to CSV or text with the `-format` modifier. For example:

```
      ]Profile data -outfile=c:\temp\data.csv -format=csv
-separators='.,'
```

creates a CSV file using a period as the decimal separator and a comma as the field separator. For more information on the `-outfile` modifier, see *Chapter 5*.

If output is directed to an XML or text file, then the `-title` modifier can be used to specify a title that will be displayed when viewing that file in the Dashboard:

```
      ]Profile tree -expr="queens 8" -title="queens eight"
-outfile="c:\temp\q8profile.xml"
```

If the `-title` modifier is omitted then the specified expression is used as the title.

The `-infile` modifier loads a previously-saved dataset for analysis – specifying this does not destroy any existing `⎕PROFILE` data:

```
      ]Profile -infile="c:\temp\test.xml"
```

This only applies when the dataset being loaded was a tree report saved in XML format (see *Section 5.1*).

B The Dashboard

The Dashboard is only available on the Microsoft Windows operating system.

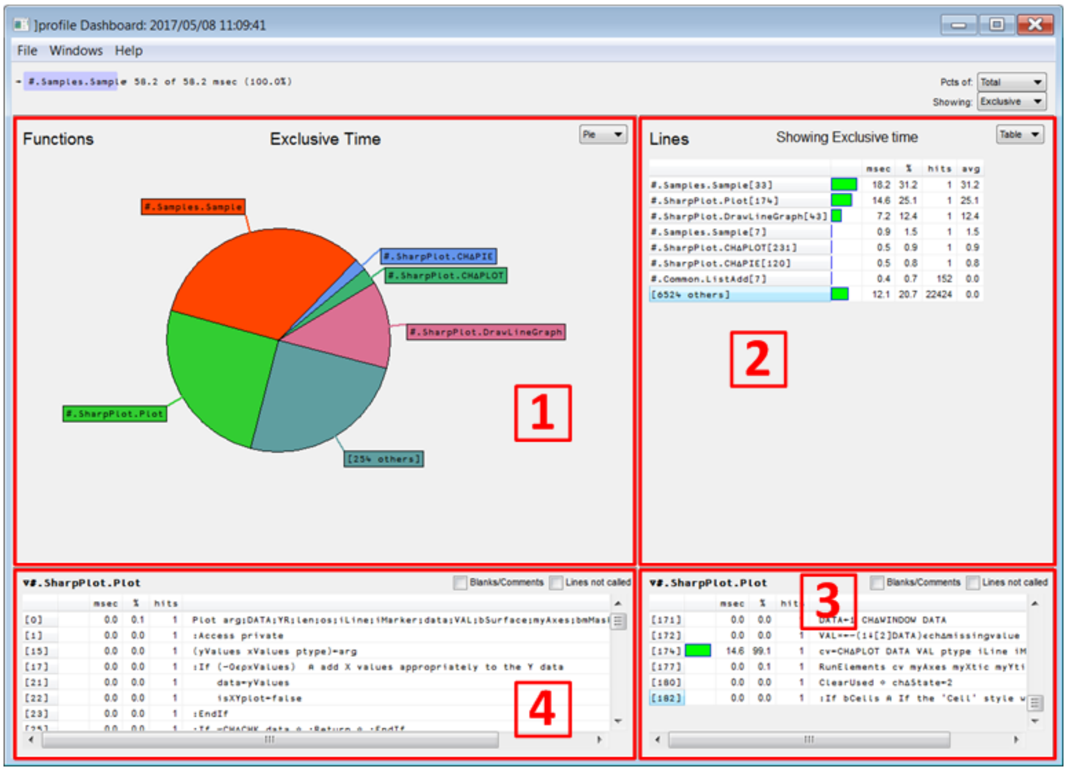
To open the Dashboard on a dataset, call the `]Profile` user command without specifying any report type, that is:

```
        ]Profile
```

The Dashboard will open and display an overview of the data currently stored by `⎕PROFILE` (`⎕PROFILE` must be stopped/inactive).

B.1 Panels

The main body of the Dashboard is divided into four panels by moveable splitters, as shown in *Figure B-1*.



```
#.Samples.Sample 58.2 of 58.2 msec (100.0%)
```

		msec	%	hits	avg
#.Samples.Sample[33]	🟩	18.2	31.2	1	31.2
#.SharpPlot.Plot[174]	🟩	14.6	25.1	1	25.1
#.SharpPlot.DrawLineGraph[43]	🟩	7.2	12.4	1	12.4
#.Samples.Sample[7]		0.9	1.5	1	1.5
#.SharpPlot.CHΔPLOT[231]		0.5	0.9	1	0.9
#.SharpPlot.CHΔPIE[120]		0.5	0.8	1	0.8
#.Common.ListAdd[7]		0.4	0.7	152	0.0
[6524 others]	🟩	12.1	20.7	22424	0.0

```
∇#.SharpPlot.Plot
         msec    %   hits
[0]       0.0  0.1     1  Plot arg;DATA;YR;len;os;iLine;iMarker;data;VAL;bSurface;myAxes;bmMas
[1]       0.0  0.0     1  :Access private
[15]      0.0  0.0     1  (yValues xValues ptype)←arg
[17]      0.0  0.0     1  :If (~0∊⍴xValues)  ⍝ add X values appropriately to the Y data
[21]      0.0  0.0     1      data←yValues
[22]      0.0  0.0     1      isXYplot←false
[23]      0.0  0.0     1  :EndIf
[25]      0.0  0.0     1  :If ~CHΔCHK data ⋄ :Return ⋄ :EndIf
```

```
∇#.SharpPlot.Plot
         msec    %   hits
[171]     0.0  0.0     1  DATA←1 CHΔWINDOW DATA
[172]     0.0  0.0     1  VAL←←~(1↓[2]DATA)∊chΔmissingvalue
[174] 🟩  14.6 99.1     1  cv←CHΔPLOT DATA VAL ptype iLine iM
[177]     0.0  0.1     1  RunElements cv myAxes myXtic myYti
[180]     0.0  0.0     1  ClearUsed ⋄ chΔState←2
[182]     0.0  0.0     1  :If bCells ⍝ If the 'Cell' style w
```

Figure B-1: *Dashboard with moveable splitters in their default positions*

The panels shown in *Figure B-1* are:

- panel 1 – Functions panel
  Consumption broken down by function. Displayed as a pie chart by default, but can be displayed as a table using the drop-down selector in the top right corner.
- panel 2 – Lines panel
  Consumption broken down by line. Displayed as a table with lines presented in order of decreasing CPU percentage consumed, but can be displayed as a pie chart using the drop-down selector in the top right corner.
- panel 3 – Line details panel
  Only populated when a line is clicked in panel 2; displays the code of the function in which the selected line appears.
- panel 4 – Function details panel
  Only populated when a function is clicked in panel 1; displays the code of the selected function.

In *Figure B-1*, the Function details panel was populated by clicking on the pie segment for `#.SharpPlot.Plot` in the Functions panel and the Line Details panel was populated by clicking on the row for `#.SharpPlot.Plot[174]` in the table in the Lines panel.

B.2 Display Options

The information presented in the four panels can be configured using the options described in this section.

Immediately above the Lines panel are two drop-down lists:

- **Pcts of** – how the percentages listed in tables and as labels on pie charts are computed:
    - Total: The given percentage is the percentage of overall consumption. This is the default.
    - Selection: The given percentage is the percentage of consumption of the function currently being displayed.
- **Showing** – whether tables report time consumed inclusive or exclusive of time consumed in any sub-functions called (pie charts always report exclusive time):
    - Exclusive: Show the consumption of each line or function excluding time consumed in any sub-functions called. This is the default.
    - Inclusive: Show the consumption of each line or function including time consumed in any sub-functions called.

Changing the selections in these drop-down lists changes the display in the Functions panel and Lines panel.

The Functions panel and Lines panel each have a drop-down list in the top-right corner:

- **Table** – if selected, functions/lines are displayed in tabular form. Left-clicking a row in a table displays information related to that row's function/line in the Function details/Line details panel. This is the default for the Lines panel.
- **Pie** – if selected, functions/lines are displayed in a pie chart with segment sizes related to CPU percentage consumed. Left-clicking a segment in a pie chart displays information related to that segment's function/line in the Function details/Line details panel. This is the default for the Functions panel.

The Function details panel and Line details panel each have two check boxes in the top-right corner:

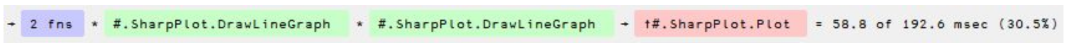
- **Blanks/comments** – if selected, the details presented will include lines that are blank or only comprise a comment. The default is for these to be omitted.
- **Lines not called** – if selected, any lines that were not called at all when running the function will be included. The default is for these to be omitted.

B.3 Navigating the Functions/Lines

A left-click in a pie segment (or on its label) or table row displays the source code for the selected function/line in the quadrant below. A double-click drills down on the relevant function/line (if possible) and updates all quadrants accordingly. Drilling down always allows indirect calls.

B.3.1 Breadcrumb Trail

Immediately above the panelled body of the Dashboard (see *Section B.1*) is a breadcrumb trail describing the function currently displayed in the panels. At the end of this breadcrumb trail is a label that reports the percentage of the overall consumption that this function accounts for. *Figure B-2* shows an example breadcrumb trail.



```
→ 2 fns  * #.SharpPlot.DrawLineGraph  * #.SharpPlot.DrawLineGraph  → ↑#.SharpPlot.Plot  = 58.8 of 192.6 msec (30.5%)
```

Figure B-2: *Example breadcrumb trail in the Dashboard*

Each breadcrumb in the trail has one of the following symbol/highlighting colour combinations:

- A right arrow (→) and blue highlighting indicate a direct call to a function without intermediate functions.
- A star (*) and green highlighting indicate a call sequence in which other functions could have been called.
- An upwards arrow (↑) and pink highlighting indicate a "show calls" step has been made, that is, consumption is displayed according to the functions/lines that have called the relevant function/line (see *Section B.3.2*).

Clicking a function in the breadcrumb trail displays that function in the panels.

B.3.2 Right-click Menu

A right-click in a pie segment (or on its label) or table row displays a pop-up menu with the following options, each of which impacts one or more panels of the display:

- **Drill Down** – Drills down one level on the relevant function/line (equivalent to double-clicking on the relevant segment/label/row). This option is only included in the pop-up menu when it is possible to drill down.
- **Make Root** – Only displays consumption that originates in the relevant function/line.
- **Show Calls** – Breaks down consumption according to the functions/lines that have called the relevant function/line (higher levels of filtering are retained).
- **Reset** – Returns to the starting position.

- **Up 1 Level** – Drills up one level (equivalent to clicking on the penultimate breadcrumb in the breadcrumb trail). This option is only included in the pop-up menu when it is possible to drill up.

B.4 Menu Bar

This section details the options available under each of the menu items in the menu bar.

B.4.1 File Menu

The options available under the **File** menu are detailed in *Table B-1*.

Table B-1: *File menu options*

Item	Description
Open	Opens an explorer window from which an XML file can be selected and analysed. Equivalent to starting the Dashboard with the `-infile` modifier set.
Save	Saves the current dataset. Equivalent to calling the `]Profile` user command with the `-outfile` modifier set.
Reset	Returns the Dashboard to its the initial state, displaying the initial top-level function in four panels (as shown in *Figure 4-1*).
Exit	Terminates the Dashboard and returns to the Dyalog Session.

For panel number references, see *Section B.1*.

B.4.2 Windows Menu

The options available under the **Windows** menu are detailed in *Table B-2*.

Table B-2: *Windows menu options*

Item	Description
Reset	Positions the vertical and horizontal splitters in their default position, as seen when first opening the Dashboard, and display the first function in the breadcrumb trail.

Table B-2: *Windows menu options (continued)*

Item	Description
Functions	Moves the vertical splitter to the right hand edge of the Dashboard, displaying only the functions panels (panels 1 and 4). This can also be achieved by double-clicking at the top of the Functions panel (panel 1).
Function Details	Moves the vertical splitter to the right-hand edge of the Dashboard and the horizontal splitter to the top of the Dashboard, displaying only the Function details panel (panel 4). This can also be achieved by double-clicking at the top of the Function details panel (panel 4).
Lines	Moves the vertical splitter to the left-hand edge of the Dashboard, displaying only the lines panels (panels 2 and 3). This can also be achieved by double-clicking at the top of the Lines panel (panel 2).
Line Details	Moves the vertical splitter to the left-hand edge of the Dashboard and the horizontal splitter to the top of the Dashboard, displaying only the Line details panel (panel 3). This can also be achieved by double-clicking at the top of the Line details panel (panel 3).

For panel number references, see *Section B.1*.

B.4.3 Help Menu

The options available under the **Help** menu are detailed in *Table B-3*.

Table B-3: *Help menu options*

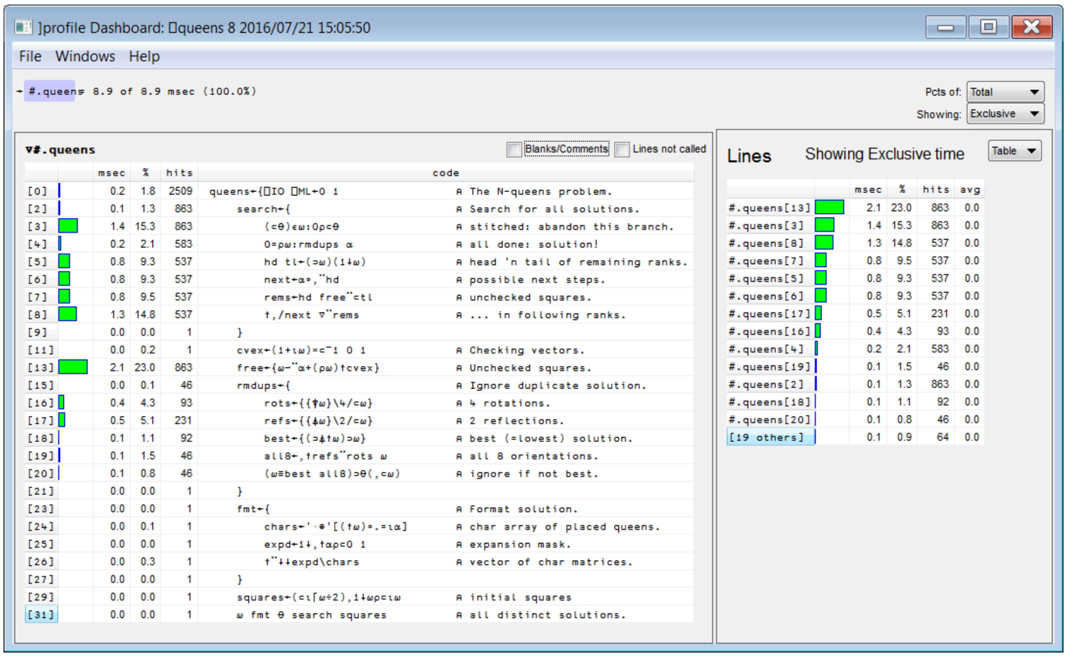
Item	Description
About	Displays the version number of the `]PROFILE` user command and the corresponding user command framework.

B.5 Single Function Mode

If the data set only pertains to a single function, then the dashboard displays two panels rather than four (as shown in *Figure B-3*). In this situation, the panel on the left displays the detailed view of the function body (equivalent to panel 3 or 4 in *Figure B-1*); the panel on the right displays the Lines panel (equivalent to panel 2 in *Figure B-1*).

EXAMPLE

```
      )LOAD dfns
      ]Profile -expr="ρqueens 8"
```



]profile Dashboard: ⎕queens 8 2016/07/21 15:05:50

File Windows Help

→ #.queens 8.9 of 8.9 msec (100.0%)

Pcts of: Total
Showing: Exclusive

∇#.queens

Blanks/Comments Lines not called

	msec	%	hits	code
[0]	0.2	1.8	2509	`queens←{⎕IO ⎕ML←0 1` ⍝ The N-queens problem.
[2]	0.1	1.3	863	`search←{` ⍝ Search for all solutions.
[3]	1.4	15.3	863	`(⊂⍬)∊⍵:0⍴⊂⍬` ⍝ stitched: abandon this branch.
[4]	0.2	2.1	583	`0=⍴⍵:rmdups ⍺` ⍝ all done: solution!
[5]	0.8	9.3	537	`hd tl←(⊃⍵)(1↓⍵)` ⍝ head 'n tail of remaining ranks.
[6]	0.8	9.3	537	`next←⍺∘,¨hd` ⍝ possible next steps.
[7]	0.8	9.5	537	`rems←hd free¨⊂tl` ⍝ unchecked squares.
[8]	1.3	14.8	537	`↑,/next ∇¨rems` ⍝ ... in following ranks.
[9]	0.0	0.0	1	`}`
[11]	0.0	0.2	1	`cvex←(1+⍳⍵)×⊂¯1 0 1` ⍝ Checking vectors.
[13]	2.1	23.0	863	`free←{⍵~¨⍺+(⍴⍵)↑cvex}` ⍝ Unchecked squares.
[15]	0.0	0.1	46	`rmdups←{` ⍝ Ignore duplicate solution.
[16]	0.4	4.3	93	`rots←{{⍒⍵}\4/⊂⍵}` ⍝ 4 rotations.
[17]	0.5	5.1	231	`refs←{{⍋⍵}\2/⊂⍵}` ⍝ 2 reflections.
[18]	0.1	1.1	92	`best←{(⊃⍋↑⍵)⊃⍵}` ⍝ best (=lowest) solution.
[19]	0.1	1.5	46	`all8←,↑refs¨rots ⍵` ⍝ all 8 orientations.
[20]	0.1	0.8	46	`(⍵≡best all8)⊃⍬(,⊂⍵)` ⍝ ignore if not best.
[21]	0.0	0.0	1	`}`
[23]	0.0	0.0	1	`fmt←{` ⍝ Format solution.
[24]	0.0	0.1	1	`chars←'·⍟'[(↑⍵)∘.=⍳⍺]` ⍝ char array of placed queens.
[25]	0.0	0.0	1	`expd←1↓,↑⍺⍴⊂0 1` ⍝ expansion mask.
[26]	0.0	0.3	1	`↑¨↓↓expd\chars` ⍝ vector of char matrices.
[27]	0.0	0.0	1	`}`
[29]	0.0	0.0	1	`squares←(⊂⍳⌈⍵÷2),1↓⍵⍴⊂⍳⍵` ⍝ initial squares
[31]	0.0	0.0	1	`⍵ fmt ⍬ search squares` ⍝ all distinct solutions.

Lines   Showing Exclusive time   Table

	msec	%	hits	avg
#.queens[13]	2.1	23.0	863	0.0
#.queens[3]	1.4	15.3	863	0.0
#.queens[8]	1.3	14.8	537	0.0
#.queens[7]	0.8	9.5	537	0.0
#.queens[5]	0.8	9.3	537	0.0
#.queens[6]	0.8	9.3	537	0.0
#.queens[17]	0.5	5.1	231	0.0
#.queens[16]	0.4	4.3	93	0.0
#.queens[4]	0.2	2.1	583	0.0
#.queens[19]	0.1	1.5	46	0.0
#.queens[2]	0.1	1.3	863	0.0
#.queens[18]	0.1	1.1	92	0.0
#.queens[20]	0.1	0.8	46	0.0
[19 others]	0.1	0.9	64	0.0

Figure B-3: *Dashboard in single function mode*