White Paper *TN002 - QtBrowser Tooling*

Modification Date   *11 June 2015*

Project   *Dawn – UPC Browser integration*

Author   *Paul von Spreckelsen*

Table of Contents

Summary

This TN describes the purpose and usage of a number of development/supporting tools as delivered by Metrological for the Dawn set-top box, being the *Web Inspector, Garbage Collector, FPS Counter, Compositing Rectangles* and the *JavaScript Memory Inspector*. These tools are part of the **qtbrowser** application as delivered by Metrological (formerly known as the TntBrowser).
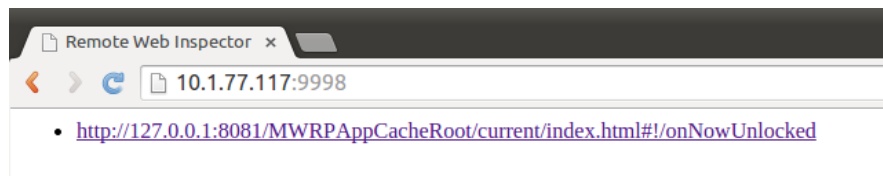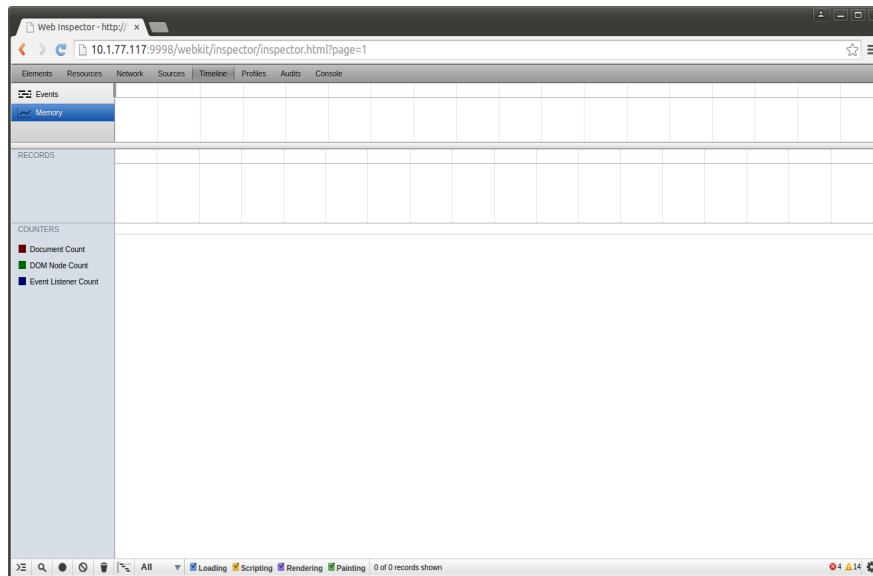
Web Inspector

Description

The Web Inspector enables a developer to view the web page source, live DOM hierarchy, script debugging, profiling and more! The Web Inspector resembles the *Inspect element* option of the Chrome browser as delivered by Google (Google Chrome Web Inspector; for more detailed information, see [1]).
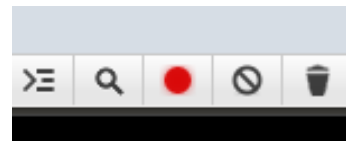
Usage

1. Start **qtbrowser** with an extra command-line parameter, being: —**inspector=<port_number>**, for example: *--inspector=9998*
2. Start the Google Chrome browser on a client PC
3. Enter the IP address of the box and the **port_number** of the web inspector in the browser URL address field, for example: *10.1.77.117:9998*. See under Tips & Tricks on how-to Obtaining IP address of box for Web Inspector.
4. A single web-page is displayed containing a hyper-link to the output page of the web inspector

5. Clicking on this link will bring up the output of the web inspector (view shown below displays the **Timeline** view of the Web Inspector)

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

6. To start/stop the web inspector press the Record icon in the status bar at the bottom of the page

Tips & Tricks

1. Please note that the Web Inspector objective is to support developers in debugging JavaScript related issues and was never mend to be an endurance tool

2. Web Inspector can only be used with the Google Chrome browser; other browsers, like Chromium or Firefox, will experience issues.

3. For logging and debugging purposes the box supports an output on an internal connector, which can be viewed by means of a serial terminal application. For Windows, a commonly used serial terminal application is PuTTy (see for example [2] on how-to configure PuTTy for serial communications); whereas for Linux a commonly used serial terminal application is minicom. See [3] for more detailed background information on serial communications. In order to view the serial output properly, the following serial communication settings need to be set:

   - speed/baud-rate: 115200
   - data bits: 8
   - parity: N(one)
   - stop bits: 1

- Software flow control No (optional, only change in case of issues)
- Hardware flow control No (optional, only change in case of issues)

4. Obtaining IP address of box for Web Inspector

- open the hood of the box and connect a serial cable on the one side to the connector inside the box (right figure), and the other side to a host PC (left figure);
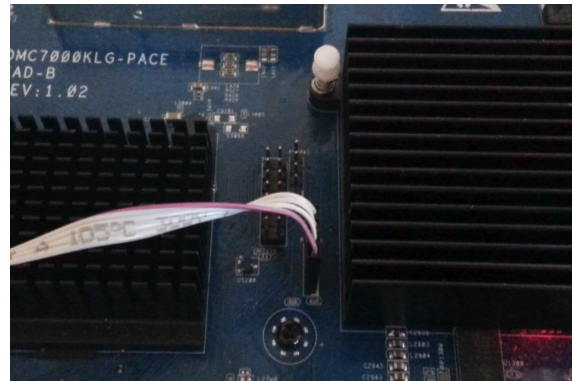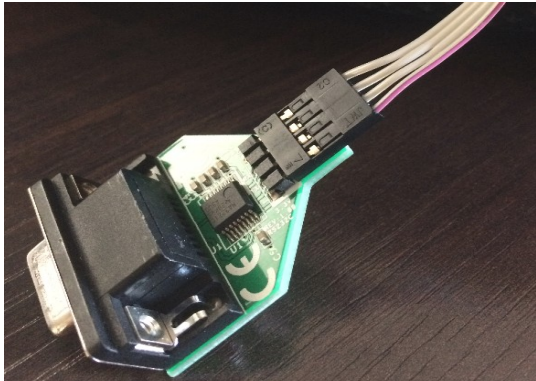
- on the host PC, open a serial terminal window (e.g., minicom, PuTTy) to the box; this serial terminal enables you to send commands remotely over the serial line to the box by means of a command-line inside the serial terminal window;

- on the command-line in the serial terminal enter: ***ifconfig eth2*** to find the IPv4 address of the box (being the inet addr, as seen in the figure below);

- the IP address obtained is the IP address to be used for the Web Inspector.


```
~ # ifconfig eth2
eth2      Link encap:Ethernet  HWaddr 00:50:B6:10:90:59
          inet addr:10.1.77.117  Bcast:10.1.255.255  Mask:255.255.0.0
          inet6 addr: fe80::250:b6ff:fe10:9059/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:122811 errors:0 dropped:4 overruns:0 frame:0
          TX packets:52132 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:151633598 (144.6 MiB)  TX bytes:8397771 (8.0 MiB)
```

5. The USB port to the side of the box can be used as an Ethernet port (being configured as **eth2**) by means of an USB-to-Ethernet cable. In the Release and Debug build, this port is enabled and not protected by any firewall. In contrast to the Ethernet ports at the back of the box which are, by default, protected by means of a firewall. Meaning that attempting to connect to the Web Inspector via the back Ethernet ports may not be possible, unless the firewall had been turned off. Note: in case of a Production build this option is disabled as well!

White Paper *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

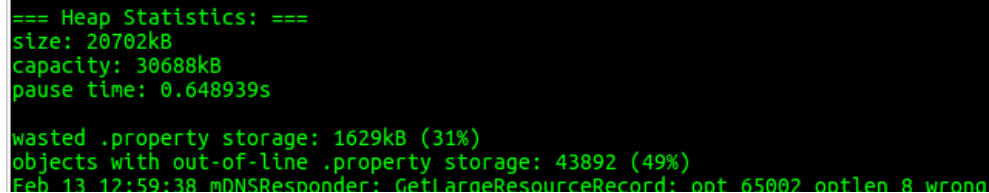# Garbage Collector Statistics

Description

Shows statistics about the garbage collection, current and total heap size, how much was cleared and how long it took the garbage collector to collect and free up the garbage.

Usage

It's usage is triggered by setting an environment variable:

> ***export JSC_showObjectStatistics=1***

Once set, the garbage collection statistics are printed to screen every 110 seconds, give or take. The output can be seen in a serial terminal window and looks like:

```
=== Heap Statistics: ===
size: 20702kB
capacity: 30688kB
pause time: 0.648939s

wasted .property storage: 1629kB (31%)
objects with out-of-line .property storage: 43892 (49%)
Feb 13 12:59:38 mDNSResponder: GetLargeResourceRecord: opt 65002 optlen 8 wrong
```

where:

- Size   : total size of objects allocated on the JS heap after last garbage collection;
- Capacity   : total size of memory allocated for the JS heap;
- Pause time   : duration of last garbage collection action;
- wasted .property storage   : freed heap memory during last garbage collection;
- objects with out-of-line   : number of objects having memory allocated on the js heap.

Tips & Tricks

-

FPS Counter
Description

Enabling this feature will result into a small box in the upper left corner displaying the frames per second counter.

Usage

This tool is enabled by setting the following environment variable:

> ***export WEBKIT_SHOW_FPS=<time interval in seconds>***.

White Paper　*TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

Tips & Tricks

1. The small box displaying the FPS Counter is located in the far upper left corner. Depending on the settings of the television (wide-screen, 3:4 ratio, and so one), the box might not be invisible. Make sure the television is not in over-scan, otherwise the box might fall off the screen.
2. The parameter *time interval in seconds* controls the refresh rate of the FPS Counter on screen. During this period the number of frames are counted and when expired will be written to screen.

Compositing Rectangles

Description

When enabled, the browser will show rectangles around the view-ports being drawn/updated on the display by the compositor. As part of the rectangular a number is shown in the upper left corner of the rectangular, indicating the number of copy actions from CPU memory to GPU memory. Typically, this number should be low (in the order of 1 or 2, depending on the operation performed on the data, i.e. moving data around on the display does not require a CPU-to-GPU memory copy, it is something that can be handled perfectly by the GPU itself).

Usage

This tool is enabled by setting the following environment variable:
　　　export WEBKIT_SHOW_COMPOSITING_DEBUG_VISUALS=1.
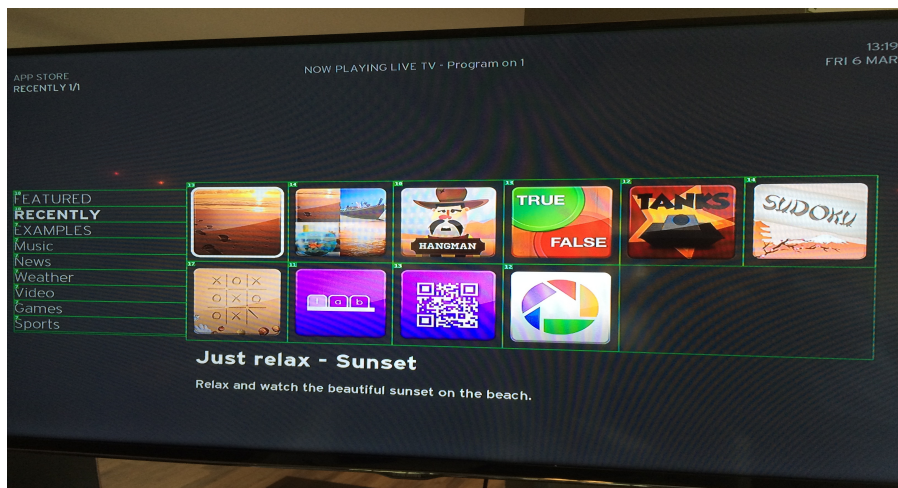
Tips & Tricks

1. Copying data from CPU memory to GPU memory are performance impacting operations and should be avoided if not required.

White Paper *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

JSMInspector

Description

The JavaScript Memory Inspector, being part of the **qt5webkit**, can be used to track RAM allocations made by JavaScript app's. In particular, it can be used to find which app allocated memory and when this memory was released, or forgotten to be released.

Usage

This tool is enabled by setting the following environment variable:
> ***export JSC_showAllocationBacktraces=1***

or, alternatively, something like:

JSC_showAllocationBacktraces=1 qtbrowser –url**=*http://example.com/ ***2> qtbrowser.memory.log

WARNING: USE THIS TOOL WITH CARE, SINCE IT WILL PRODUCE AN AWFUL LOT OF OUTPUT.

Tips & Tricks

1. A supporting script, called wkmemlogparser.scm (see Appendix A – JavaScript Inspector supporting script or [5] for the on-line version in github) was written in Guile scheme [4] to parse the log generated by the tool and give a meaningful output; it requires guile >= 2.0 to be installed. The script has various options, but the ones of interest should be:

   > ***write-backtraces***: This parses the log file and writes the parsed back-traces as S-expressions into an output file given as parameter. This output file could then be used by other scripts if you want to do some special analysis but don't like scheme. There should be S-expression parsing tools available for most major languages. Example usage:
   > ```
   > ./wkmemlogparser.scm write-backtraces qtbrowser.memory.log qtbrowser.memory.dat
   > ```

   > ***print-tree-from-log***: Given a log file generated as explained in the first section, this script will print on the standard output a "tree of allocations". Example usage:
   > ```
   > ./wkmemlogparser.scm print-tree-from-log qtbrowser.memory.log > qtbrowser.memory.tree
   > ```

2. You need to ensure that `ENABLE_JS_MEMORY_TRACKING` is defined in WTF/wtf/Platform.h, which is the default in Debug mode, but not in Release.

Best Practices

Methodology to use with web inspector tool

No long runs / endurance test !!!!

1. If core signal 11 qtbrowser is seen this means that in S99_QTBROWSER the web inspector parameter are enabled, as seen in following tickets:

   a) [#17438] [QT5] Youtube continuous playout leads to core.qtbrowser.11

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

b) [#17442] [QT5] Vodstore - open/close films/a-z/ leads to core11.qtbrowser - oom seen

c) [#17381] [QT5] core.qtbrowser signal 11 seen - 3sec zap with ongoing recording in the background

This lead to creation of below ticket:

[#17478] [QT5] webinspector needs to be disabled in release mode

=> one exception seen so far [#17461] [QT5] 8 sec channel changes leads to either core11.qtbrowser within 12 hours I was able to reproduce without web inspector active

Correct way to use web-inspector

1. Startup your box with in S99_QTBROWSER the webinspector parameters enabled

2. Open on your box already 1 once the gui view you wanna check (in this example the vodstore)

3. Start chrome web inspector session when your box is on live a/v idle for some time - here you have your baseline

4. Execute problem scenario 2 or 3 times with 10 min idle time in between

for example:

(a) rcu_vodstore

(b) Wait 10 sec

(c) Rcu_livetv

(d) Wait 10 sec

(e) => repeat this for few minutes

(f) Leave box idle to see if DOM nodes / listeners / documents drop back to baseline of step 3

How to identify which component is responsible for the leak

1. DOM nodes / listeners are only created by the JS from EPAM / Metrological, SCIH NAF team is not creating any DOM nodes / listeners

a) There is a way to check if it's EPAM or Metrological => on the 'elements' tab delete the iframe line of Metrological and run the same use-case

2. If no DOM / nodes / documents are increasing and browser is still going out of memory we need to look at SCIH NAF use-cases => mrecord stats is the only way to verify it…

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

References

[1]   https://developer.chrome.com/devtools/docs/timeline

[2]   http://kb.cyberoam.com/default.asp?id=2193

[3]   http://en.wikipedia.org/wiki/Serial_port

[4]   https://www.gnu.org/software/guile/guile.html

[5]   https://github.com/guijemont/wkmemlogparser

Appendix A – JavaScript Inspector supporting script

wkmemlogparser.scm

```
#!/usr/bin/guile-2.0 \
-e main -s
!#

(use-modules (ice-9 rdelim))
(use-modules (ice-9 match))
(use-modules (ice-9 receive))
(use-modules (srfi srfi-1))
(use-modules (srfi srfi-9 gnu))
(use-modules (srfi srfi-26))

(define (tokenize)
(let ((line (read-line)))
(if (eof-object? line)
line
(string-split line #\space))))

(define (backtrace address size frames) `(backtrace ,address ,size ,frames))

(define (display-backtrace backtrace)
(match backtrace
(('backtrace address size frames)
(format #t "~A: ~A bytes\n" address size)
(for-each (cut format #t "~A\n" <>) frames)))))
```

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

```scheme
(define (parse-fold proc seed)
(define (index? str)
(string-prefix? "#" str))

(let lp ((current-address #f)
(current-size #f)
(current-frames '())
(tokens (tokenize))
(seed seed))
(match tokens
((? eof-object? eof) seed)

((size "bytes" "at" address)
(lp address size '() (tokenize)
(if current-address
(proc (backtrace current-address current-size (reverse current-frames)) seed)
seed)))

(((? index? _) function "at" location)
(lp current-address current-size
(cons (cons function location) current-frames) (tokenize) seed))

(("Backtrace" "end")
(let ((new-seed (proc
(backtrace current-address current-size (reverse current-frames))
seed)))
(lp #f #f '() (tokenize) new-seed)))

(((or "[GC:" "FullCollection," "EddenCollection,") args ...)
; these messages tend to be "out of line", so we "eat" them here
(lp current-address current-size current-frames args seed))

(_ ; ignore everything else
(lp current-address current-size current-frames (tokenize) seed)))))

(define (parse backtrace-action)
(parse-fold
(lambda (bt seed) (backtrace-action bt))
#f))

(define (fold-backtraces proc seed)
```

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

```
(let lp ((seed seed))
(let ((backtrace (read)))
(if (eof-object? backtrace)
seed
(lp (proc backtrace seed))))))

(define (parse-dat backtrace-action)
(fold-backtraces
(lambda (backtrace seed) (backtrace-action backtrace))
#f))

; (vector node location total hits children)
(define (insert-backtrace root reversed-frames alloc-size)
"For a backtrace represented by REVERSED-FRAMES (frames from bottom to top), insert
it in the children of the tree ROOT, considering that it did allocate
ALLOC-SIZE.
This procedure is not tail-recursive, but you shouldn't have that many frames
anyway, right?"
(define (node location total hits children) (vector 'node location total hits children))

(define (new-subtree frames)
(match frames
(() '())
((bottom-frame other-frames ...)
(list (node bottom-frame alloc-size 1 (new-subtree other-frames))))))

(define (find-matching-child loc children)
(define (valid-child? child)
(equal? (vector-ref child 1) loc))

(receive (matching other-children)
(partition valid-child? children)
(match matching
(() (values #f children))
((matching-child) (values matching-child other-children))
(_ (error "More than one child with matching location!")))))

(match root

(#('node loc total hits ()) ; no children
(node loc (+ total alloc-size) (1+ hits) (new-subtree reversed-frames)))
```

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

```
(#('node loc total hits children)
(match reversed-frames
(() (node loc total hits children))
((bottom-frame other-frames ...)
(node loc (+ total alloc-size) (1+ hits)
(receive (child more-children)
(find-matching-child bottom-frame children)
(if child
(cons (insert-backtrace child other-frames alloc-size) more-children)
(append (new-subtree reversed-frames) children)))))))))))

(define (make-tree fold-proc)
(fold-proc
(lambda (backtrace root)
(match backtrace
(('backtrace _ sizestr frames)
(insert-backtrace root (reverse frames) (string->number sizestr)))))
(vector 'node #f 0 0 '())))

(define (print-tree root)
(define (compare-nodes node1 node2)
(match (cons node1 node2)
((#('node _ total1 _ _) . #('node _ total2 _ _))
(>= total1 total2))))

(define (print-node node depth)
(let print-prefix ((depth depth))
(cond
((> depth 1)
(display "| ")
(print-prefix (1- depth)))
((equal? depth 1)
(display "->"))
(else (error "Invalid depth in print-prefix"))))
(match node
(#('node (function . loc) total hits _)
(format #t "~A bytes in ~A calls ~A (~A)\n" total hits function loc)))
```

White Paper   *TN002 - QtBrowser Tooling*

Modification Date	*11 June 2015*
Project	*Dawn – UPC Browser integration*
Author	*Paul von Spreckelsen*

```
(let lp ((root root) (depth 0))
(match root
(#('node #f total hits children) (format #t "Total allocations: ~A bytes\n" total))
(node (print-node node depth)))

(match root
(#('node _ _ _ children)
(for-each
(lambda (child) (lp child (1+ depth)))
(sort children compare-nodes))))))

(define (main args)
(match args

((_ "show-backtraces" filename)
(with-input-from-file filename (lambda () (parse display-backtrace))))

((_ "write-backtraces" logfile outfile)
(with-output-to-file outfile
(lambda ()
(with-input-from-file logfile (lambda () (parse write))))))

((_ "show-data" datafile)
(with-input-from-file datafile (lambda () (parse-dat display-backtrace))))

((_ "print-tree" datafile)
(with-input-from-file datafile (lambda () (print-tree (make-tree fold-backtraces)))))

((_ "print-tree-from-log" logfile)
(with-input-from-file logfile (lambda () (print-tree (make-tree parse-fold)))))

((cl . _)
(format (current-error-port) "usage ~A COMMAND ARG...\n" cl)
(format (current-error-port) "Available commands\n")
(format (current-error-port) " show-backtraces LOGFILE\n")
(format (current-error-port) " write-backtraces LOGFILE OUTDATAFILE\n")
(format (current-error-port) " show-data DATAFILE\n")
(format (current-error-port) " print-tree DATAFILE\n")
(format (current-error-port) " print-tree-from-log LOGFILE\n")
(exit 1))))
```