Internship Report

INTEGRATING A JAVASCRIPT ENGINE INTO A LIGHT WEB BROWSER

Internship tutor
Yann-Gaël Guéhéneuc
Associate Professor
Ptidej Team

Polytech Paris UPMC tutor
Yann Douze
Associate Professor of Electronics
Université Pierre et Marie Curie

Niverthan PANCHALINGAM
2018 PROMOTION AT POLYTECH PARIS UPMC | 4TH YEAR IN EMBEDDED SYSTEMS DEPARTMENT

Table des matières

Acknowledgments

First of all, I would like to thank the teaching staff that provided the Polytech courses. Then, of course, Mr. Yann-Gaël Guéhéneuc, a professor at Polytechnique Montréal and my tutor, for having welcomed me very warmly to his team, and to have ensured day after day the good of my internship. I also thank him for the sharing of his knowledge on a daily basis. Finally, I would like to thank all the people I have worked with for their contribution to my integration and the sympathy they have shown me during this internship, as well as Polytech Paris UPMC and Ecole Polytechnique de Montréal for allowing me to do it.

Polytechnique Montréal

I did my 3-month internship at École Polytechnique de Montréal, in the Ptidej team. It is situated in the Université de Montréal, and is split into several departments like Computer Engineering, Electrical Engineering, …

Polytechnique Montréal is composed of two buildings: the main one, where courses are given, and the second one, the Pavillon Lassonde, where the research labs are located. My internship was carried into the second building, since it holds the Computer Engineering department lab, where my tutor, Yann-Gaël Guéhéneuc, is working.

Among the several research teams present in the Computer Engineering department, I worked in the Ptidej team (Pattern Trace Identification, Detection, and Enhancement in Java), directed by Mr. Guéhéneuc.

Even if NetSurf isn't coded in Java, it's this browser that my tutor had decided to make me work on during my internship. It was the continuation of two previous internships, with the last one being done by Dylan Levy, another IG student from Polytech Montpellier.

I. Introduction

1. Mission

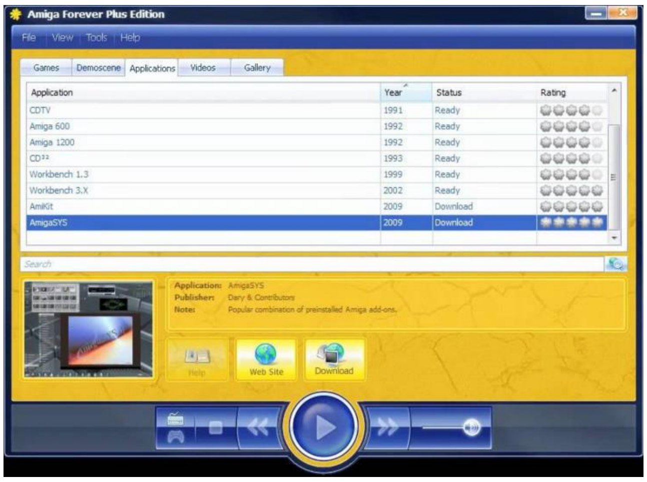
My mission was to integrate a JavaScript engine (named DukTape) to the browser NetSurf. A previous work on this project was made by a Polytech Montpellier student, Dylan Levy, focusing on the implementation of the browser NetSurf on AmigaOS3 via the creation of a script that compiles NetSurf to run on AmigaOS. It was done in summer 2016. However, when compiled with DukTape (the JavaScript engine integrated to NetSurf), not only does the browser crash, but it also makes the whole OS crash. So my mission was first to examine and solve this problem, and then to integrate DukTape into NetSurf.

2. AmigaOS, Amiga Forever & Amikit

AmigaOS is a family of operating systems of the Amiga and the AmigaOne personal computers. It was introduced with the launch of the first Amiga, the Amiga 1000, in 1985.

AmigaOS is a single-user operating system based on a preemptive multitasking kernel, called Exec. It includes an abstraction of the Amiga's hardware, a disk operating system called AmigaDOS, a windowing system API called Intuition and a desktop file manager called Workbench.

Since I had no computer running on AmigaOS, I used the official AmigaOS emulator: Amiga Forever, and made it run on a Windows 10 pc, on which I was working.



Application	Year	Status	Rating
CDTV	1991	Ready	●●●●○
Amiga 600	1992	Ready	●●●●○
Amiga 1200	1992	Ready	●●●●○
CD^{32}	1993	Ready	●●●●○
Workbench 1.3	1999	Ready	●●●●○
Workbench 3.X	2002	Ready	●●●●●
AmiKit	2009	Download	●●●●●
AmigaSYS	2009	Download	●●●●●

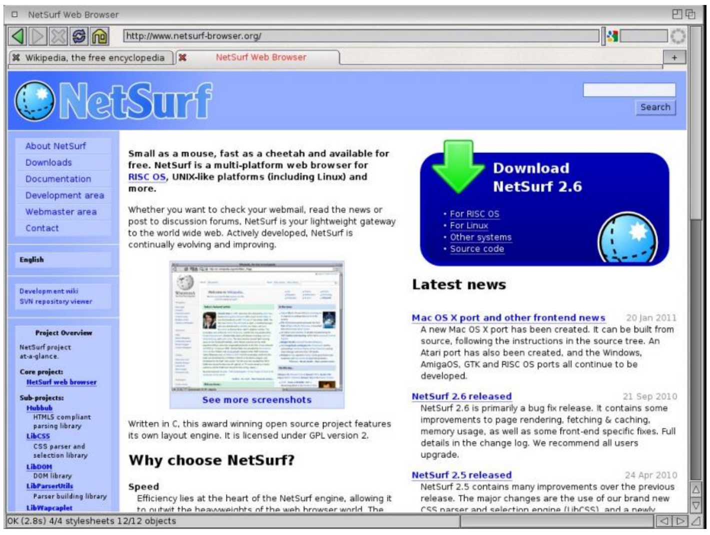
Application: AmigaSYS
Publisher: Dary & Contributors
Note: Popular combination of preinstalled Amiga add-ons.

It allows Amiga software to run on non-Amiga hardware legally and without complex configuration. However, you have to buy a key to use it, but Mr Guéhéneuc lend me one.

Amiga Forever allows the use of Amikit very simply. It's a preconfigured package of more than 350 Amiga programs running on Windows, Mac or Linux computers.

3. NetSurf

NetSurf is a free and open source browser, written in C/C++. Netsurf has its own layout and rendering engine entirely written from scratch. It is small and capable of handling many of the web standards in use today. Since AmigaOS doesn't have any good working browser, the goal of Dylan Levy's internship of last year was to make NetSurf run on AmigaOS. And my goal was to integrate DukTape into it, to allow JavaScript execution.



Project Overview

NetSurf project at-a-glance.

Core project:
NetSurf web browser

Sub-projects:
Hubbub
HTML5 compliant parsing library
LibCSS
CSS parser and selection library
LibDOM
DOM library
LibParserUtils
Parser building library
LibWapcaplet

Small as a mouse, fast as a cheetah and available for free. NetSurf is a multi-platform web browser for RISC OS, UNIX-like platforms (including Linux) and more.

Whether you want to check your webmail, read the news or post to discussion forums, NetSurf is your lightweight gateway to the world wide web. Actively developed, NetSurf is continually evolving and improving.

Written in C, this award winning open source project features its own layout engine. It is licensed under GPL version 2.

Why choose NetSurf?

Speed
Efficiency lies at the heart of the NetSurf engine, allowing it to outwit the heavyweights of the web browser world. The

Latest news

Mac OS X port and other frontend news 20 Jan 2011
A new Mac OS X port has been created. It can be built from source, following the instructions in the source tree. An Atari port has also been created, and the Windows, AmigaOS, GTK and RISC OS ports all continue to be developed.

NetSurf 2.6 released 21 Sep 2010
NetSurf 2.6 is primarily a bug fix release. It contains some improvements to page rendering, fetching & caching, memory usage, as well as some front-end specific fixes. Full details in the change log. We recommend all users upgrade.

NetSurf 2.5 released 24 Apr 2010
NetSurf 2.5 contains many improvements over the previous release. The major changes are the use of our brand new CSS parser and selection engine (LibCSS), and a newly

OK (2.8s) 4/4 stylesheets 12/12 objects

4. NetScript & Cygwin

NetScript is a script created by the previous trainee, Dylan Levy, in 2016. Its main goal is to automate the NetSurf compilation process and make it user and developer-friendly. It evolved continuously during the internship. It is composed of 4 main files and folders :

- NS.sh : this file contains the commands to download and compile all the files, tools and libraries needed to compile NetSurf
- NetScript.sh : this file is used to execute NetScript. It only calls NS.sh but allows NetScript to create a log file (in case the NetSurf compilation does not work)
- updateFiles.sh : this file is in charge of modifying all the files needed to compile NetSurf (for example, the NetSurf Makefile)
- the updateFile folder contains data to modify in each file in order to enable the NetSurf compilation.

At NetScript launch, one has several options to alter the compilation. For example, you can compile NetSurf with or without DukTape, keep the files and libraries used during the compilation (to dig in the code), or even keep the /opt/netsurf folder (which contains the tools used to compile NetSurf). Finally, you can choose to clean its workspace before the script execution, to begin from scratch.

I altered NetScript to best suit my needs: to automate the process even further, and do other things such as closing and relaunching Amiga Forever.

Cygwin is a Shell emulator based on Unix/Linux systems for Windows. It is compulsory to run NetScript on a Windows computer, so I had to install it.

Cygwin

5. DukTape

DukTape is an embeddable JavaScript engine. It was the one chosen to be integrated with NetSurf, because of its focus on portability and compact footprint.

II. Launching of the project

1. Setting up the environment

As in every development project, the first step is to prepare the working environment. During the first week, there was no computer available, so I brought my laptop. The problem was that, since it is an old computer, it took a long time to make a single action. So when I had the chance to get a desktop computer, I took it. However, this made me install everything I needed a second time.

The first thing I needed was an AmigaOS emulator. My Internship tutor, M. Guéhéneuc, had a key for Amiga Forever, so it's the emulator that I used during the project. I also used Amikit, that can be downloaded from the Amiga Forever interface. With these, I had a working AmigaOS emulator.

Then, I obviously needed NetSurf, the browser I had to work on. To get it, all I had to do was to get NetScript, and make it work. So I got it from Dylan Levy's GitHub, and launched it. It soon appeared that I needed Cygwin to use it, since I was on a computer under the Windows 10 OS. To make everything clear, I had to make NetSurf work on AmigaOS, but I was working under Windows, and not AmigaOS. I downloaded and installed cygwin 64-bits, and tried launching the script again. M. Guéhéneuc and I were expecting it to not work properly after one year, and it indeed failed to compile NetSurf. There were a lot of errors, so I tried to figure out how to fix them. In the end, the problem wasn't with the script itself, but with the version of Cygwin I was using. For some reason, the script wasn't working with the 64-bits version, while it worked fine when I tried with the 32-bits one.

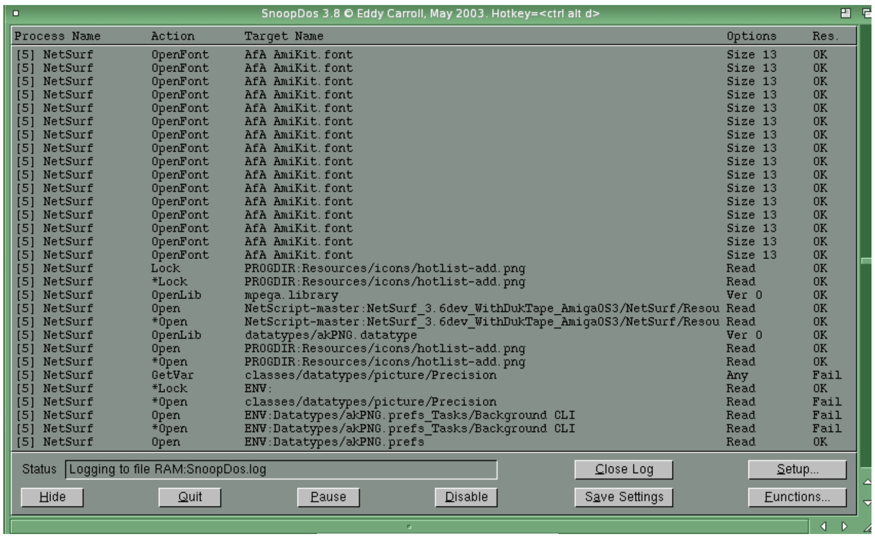
At this point, all I needed was a way to make windows and AmigaOS share files. Luckily, Amiga Forever has this feature, so it wasn't a problem.

This means that I had everything set up and ready to go.

2. Getting used to NetSurf & AmigaOS

When I finally managed to make NetScript work, I soon encountered the problem Dylan Levy (the student who worked on it last year) was facing : when compiling NetSurf with Duktape, the browser crashed at launch. In fact, it made the whole OS crash. I decided to get used to the browser before attempting to fix this problem. So I compiled it without DukTape, and browsed the web. One of the first problems I faced was about NetSurf settings. For some reason, changing the settings made me unable to change them again before the next NetSurf launch. This bug was present during all the project.

To try and make NetSurf work with DukTape, I installed SnoopDos on AmigaOS. It's a tool that can log every system call. I then figured out how to make it save its output in a file, so I could see what was happening before the crash of the OS. However, it was really hard to understand what was the crash cause, because it was logging every system call, which could be thousands in a matter of seconds. This made a lot of lines to examine.



```
Process Name        Action       Target Name                                                      Options    Res.
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         OpenFont     AfA AmiKit.font                                                  Size 13    OK
[5] NetSurf         Lock         PROGDIR:Resources/icons/hotlist-add.png                          Read       OK
[5] NetSurf         *Lock        PROGDIR:Resources/icons/hotlist-add.png                          Read       OK
[5] NetSurf         OpenLib      mpega.library                                                    Ver 0      OK
[5] NetSurf         Open         NetScript-master:NetSurf_3.6dev_WithDukTape_AmigaOS3/NetSurf/Resou Read     OK
[5] NetSurf         *Open        NetScript-master:NetSurf_3.6dev_WithDukTape_AmigaOS3/NetSurf/Resou Read     OK
[5] NetSurf         OpenLib      datatypes/akPNG.datatype                                         Ver 0      OK
[5] NetSurf         Open         PROGDIR:Resources/icons/hotlist-add.png                          Read       OK
[5] NetSurf         *Open        PROGDIR:Resources/icons/hotlist-add.png                          Read       OK
[5] NetSurf         GetVar       classes/datatypes/picture/Precision                              Any        Fail
[5] NetSurf         *Lock        ENV:                                                             Read       OK
[5] NetSurf         *Open        classes/datatypes/picture/Precision                              Read       Fail
[5] NetSurf         Open         ENV:Datatypes/akPNG.prefs_Tasks/Background CLI                   Read       Fail
[5] NetSurf         *Open        ENV:Datatypes/akPNG.prefs_Tasks/Background CLI                   Read       Fail
[5] NetSurf         Open         ENV:Datatypes/akPNG.prefs                                        Read       OK

Status  Logging to file RAM:SnoopDos.log
```

SnoopDos execution

In the end, it was luckily not needed, because shortly after the installation of this tool, there was some updates on Amiga Forever. They fixed the problem of NetSurf and DukTape : it was not crashing anymore. I could then activate JavaScript on NetSurf and see how well it was working.

In fact, it was not working very well. NetSurf was crashing a lot, and JavaScript didn't seem to work at all. This was because DukTape wasn't - and still isn't - fully integrated to NetSurf.

The goal of my internship was to figure out why NetSurf made AmigaOS crash when compiled with DukTape, and fix it. Since it was already done, M. Guéhéneuc and I decided that I should try and continue integrating DukTape to NetSurf.

3. NetSurf & JavaScript

Before beginning to code, I had to see what was already implemented, and how. In fact, a lot of methods were already in place, and supposed to work. But since I had close to no experience with JavaScript, I didn't know, nor understand much of them. So I decided to focus on two similar methods: document.getElementsByTagName, and document.getElementsByClassName. The first of them is supposed to get all elements of the page that has a specific tag, and was in fact supposed to be working. The second also get all elements of a page, but with the class name(s) given as parameter. This method was not implemented, so I decided to begin the implementation with this one, since it is very similar to getElementsByTagName.

To see whether or not some methods were working (typically, console.log()), I had to launch NetSurf in verbose mode. To do this, all I had to do was to find the AmigaOS terminal, and launch NetSurf with the "-v" option. However, I had to figure that out by myself, since it isn't stated anywhere in the NetSurf readme, or on the GitHub.

4. Setting up a testing Website

To try and see if the implemented methods were working, I had to create a little website to do basic tests. It later allowed me to test my own implementations. I chose to not make it local, so I could easily see the differences between NetSurf on AmigaOS and other browsers, such as Google Chrome or Mozilla Firefox on Windows, at the same time. I created a heroku project, and started testing the getElementsByTagName method. Heroku is a website that can be used to host websites in PHP, for example.

III. Altering Dylan Levy’s work

1. Automating the process

The whole process of compiling NetSurf takes a really long time. Dylan Levy thought about it, and made an option to make it faster (from around 20 minutes to “only” 5). To compile NetSurf with this option, NetSurf had to be compiled at least once before, because it needs to already have all the libraries present. In fact, this option doesn’t compile them to make the process go faster, and instead uses the compiled files from previous execution of the script. This was a good idea, but didn’t suit my needs, because I needed to be able to modify the libraries and compile NetSurf with these changes.

It made me code my own option, which allowed me to compile NetSurf, with only the libraries I needed. I called it the “fast” option. To use it, all I had to do was to call “NetScript.sh -f” from a terminal. And to choose the libraries, I only needed to add, for example, “--libdom”, or “-dom”, for the libdom library.

Another difference between my option and the regular one is that removed the possibility to compile NetSurf without DukTape. I didn’t need this choice anymore, since all I would be doing from this point would need DukTape.

2. Adding functionalities

This script was working fine, but still didn’t cover the whole process. In fact, after compiling NetSurf, I had to extract the resulting archive in the right location (the folder shared with AmigaOS). This also forced me to stop the emulator before, since the folder is otherwise considered “in use”, which makes it unable to be deleted/altered. I then had to re-launch Amikit, find the terminal in AmigaOS, and launch NetSurf in verbose mode. This, again, took some time to do, and developers are supposed to be lazy to be efficient. So I again altered the script to automate things the most I could.

a. The archive extraction

The first goal was to extract the archive into the shared folder. But to do that, I had to make sure that Amikit was closed. The solution was simple: shut down the Amikit process. I did this in 2 steps: first, trying to shut it down by sending it a stop signal (taskkill). If, after several seconds, the process was still running, it would shut it down by force (taskkill /F: the windows “equivalent” of the famous “kill -9” of Linux). The destination folder is then emptied, and the NetSurf archive is extracted there.

b. Default settings

The most annoying thing to do each time I had to recompile NetSurf was to choose the NetSurf window resolution, activate Javascript, and set the homepage (to the webpage I used for testing). These actions are done when NetSurf is launched, by changing its parameters. However, as I described before, changing parameters in NetSurf is a bit messy. So I wanted to automate this too. To do this, I used a directory comparison software (Meld) to see which file(s) were altered when I changed the settings in NetSurf. After finding this file, all I had to do was to modify the script to automatically create this file, with the informations needed, at the end of NetSurf compilation.

c. Launching NetSurf

The third step was the logical continuation of the first one. Since I closed Amikit before extracting the archive, I could launch it again after this action was over.

After implementing this, the script was compiling NetSurf, closing Amikit, extracting NetSurf into the folder shared with Amikit, launching Amikit again, and finally setting NetSurf settings so it had a default homepage, Javascript activated, and a default resolution set.

I chose to do one last thing. As I always needed to launch NetSurf in verbose mode, I had to open a terminal, change the current directory to the NetSurf one, launch NetSurf in verbose mode and redirect the output to the desired file. This took some time, especially finding the terminal, which was “hidden” deep in AmigaOS folders. To speed this up, I first created a file that could be executed in AmigaOS, that launched NetSurf into verbose mode and redirected its output to the file I wanted. Then, I had to figure out how to launch the terminal quickly, as creating a shortcut in the desktop wasn’t working (it was deleted at each reboot). The solution I found was to alter one of AmigaOS files, which is executed when AmigaOS is booting. It allowed me to launch a terminal window, and set its current directory to the NetSurf one. The only thing I had to do was to type “execute scriptname” to launch NetSurf. This last step couldn’t be executed automatically, because, when the file is executed by AmigaOS, the system is not fully ready yet. It can open a terminal, but not launch NetSurf at this point.

So there I had it, a script that did everything I needed: compile NetSurf with only the libraries I needed, extract it at the desired location, set its parameters, and launch Amikit. And Amikit was showing me a terminal window, in which all I had to do was to type “execute scriptname”. This allowed to really speed up the whole process.

Even with all these steps, the compilation still takes a long time: around 3-5 minutes. This means that, for each test of my code, I had to “wait” 5 minutes before actually seeing the effects of it. This was probably the most annoying thing during the project, since it made me lose a lot of time.

3. Cleaning the code

NetScript is separated in two files: NetScript.sh and NS.sh. The only purpose of NetScript.sh is to call NS.sh, and store its result in a file. It was doing it by checking the second argument. If it was "-q" or "--quick", it called NS.sh -q. Otherwise, it only called NS.sh. The whole compilation process is done in NS.sh.

a. Arguments management

Since I had to add another possible argument, -f (or --fast), I had to change this part. In fact, I found it interesting to also implement a "-h", or "--help", to explain what the script is doing. So, instead of checking the second argument value, and call NS.sh with the same argument, I decided to change NetScript.sh: all he does now is to call NetScript.sh with all the arguments that he received, and log its output into a file.

Then, I modified NS.sh to take these arguments into account. Here again, since it only had one option (-q), the script only checked if the second argument was "-q", and reacted appropriately. This was not possible anymore, so I decided to separate the script in two parts: arguments parsing, and the script itself.

First, the script parses every argument he received, and, for each, he sets a variable. For example, if he finds "-q", then he'll set the "quick" variable to 1. Then, depending on the variables values, he'll behave appropriately. Not only is this way easier to understand, but it is also easier to maintain, and to add other arguments, if needed.

b. Functions

Since I added a new option for the script, I had to code the behaviour of this option. But because I had to use a lot of code that was present in the default script, I decided to create functions, to avoid any code duplication. Again, this allows an easier code maintenance. For example, I now had function like "compile_netsurf", "good_ending", …

IV. getElementsByClassName integration

My first goal after successfully launching NetSurf with DukTape, was to see how the getElementsByTagName method worked, since it is really similar to the method I wanted to implement : getElementsByClassName.

1. getElementsByTagName testing

At first glance, this method seemed to work fine. To test it, I did a simple html page, which contained a "p" element. I also had a CSS page, which defined the class "blue". All it did was to turn the text color of the elements having this class blue. Then, there was a little piece of JavaScript:

document.getElementsByTagName(“p”).item(0).className = “blue”; (very simple, it changes the class of the element to blue). When launching NetSurf on this page, it worked fine: the text in the “p” element turned blue.

However, I soon noticed that, if the selected element already had a class, the method didn’t seem to work. For example, if I had a class to my “p” element, say “toblue”, with the same piece of JavaScript, and opened NetSurf, the text wasn’t blue. With M. Guéhéneuc, we thought that it may be because of the CSS interpreter, that may stop scanning for code change at some point. We figured out way later (after finishing implementing the getElementsByClassName method) that it was in fact due to the classes cache that wasn’t updated.

2. Trying to log

During the whole project, the function I used the most was the “LOG” function. It allowed me, as its name indicates, to log anything I wanted, when launching NetSurf in verbose mode, into the output file I chose. Understanding getElementsByTagName was made possible mainly because of this function.

The problem is that it was a function callable only from the DukTape implementation. That is, I couldn’t use it with external libraries, which I needed to alter a lot. In fact, libdom, for example, is a library that is being developed to implement the DOM to be used with NetSurf. As it is currently under development, I could alter it, and then make a pull request on their GitHub.

The best solution I found was to create global variables in libdom, that are also accessible from DukTape. That way, all I had to do was to modify these variables as I wanted in libdom, then log their content with DukTape.

3. Some problems

As I began to implement the getElementsByClassName method, I ran into several problems, which were mainly caused by my lack of C++ knowledge. For example, I had a lot of trouble dealing with structs, especially considering the difference between the declaration and the definition of them. In fact, some structures in libdom were usable outside of the library, while others were not. This made things harder to understand; and logging the content of these variables was tricky too.

Another unusual aspect was the declaration of the methods. In fact, most of them were preceded by an underscore (_) (for example: _get_elements_by_tag_name). However, when used outside of libdom, they could be used without the underscore (example: get_elements_by_tag_name). Moreover, calling them with or without the underscore seemed to deliver different outputs, or even not work at all. I never managed to find how the underscores were altering the process, but having it or not implied calling different methods from different files/classes. For example, dom_document_get_elements_by_tag_name was calling this function from the html_document.c file, while _dom_document_get_elements_by_tag_name was calling the same function, but from the document.c file. Both had the same name: _dom_document_get_elements_by_tag_name.

4. Understanding getElementsByTagName

To understand the getElementsByTagName method, I followed the functions it is calling. The main function is called "_dom_document_get_nodelist", so I tried to understand it. Given its name, I supposed it was supposed to get a nodelist (which is a list of nodes, which are basically elements of a webpage). However, that's not exactly what it does. In fact, it creates all the arguments that define this nodelist, and stores them. For example, with getElementsByTagName, it only stores the tag name given as argument. But it doesn't browse the dom to find the elements that correspond to such arguments. This allows a really low memory impact - which is the main goal of NetSurf - but was hard to understand. It is called "lazy evaluation", and it's something I never saw before.

The DOM is browsed only when searching for an item of this list (for example, if we want the second item, it will browse the DOM until it finds the second node that has corresponding arguments, here the right tag name), or when the "length" property is needed (even if length is a property, it is defined as a function in libdom).

The downside of this is obviously its speed. Every time we want to know the length of a nodelist, we need to browse the whole DOM to find it out.

5. Implementation: a mitigated success

To implement the getElementsByClassName method, I had to alter a bit the nodelist defined in libdom, by adding a "class" property inside its structure. I also had to define a new way of searching: BY_CLASS_NAME.

But this wasn't enough, since the "item" and "length" methods of the nodelist class weren't defined for such a nodelist! I altered these two methods to take it into account. Luckily, there already were a function from the element.c file, that can check whether or not this element has a given class. So this was the only test that was done: if an element has the given class, then it is part of the nodelist.

Finally, I could call the "_dom_document_get_nodelist" function inside my

dom_document_get_elements_by_class_name method, to get the nodelist corresponding to all the elements that have a given class name.

However, there were two problems with this implementation.

The first one was visible: when trying to see if my method worked, I tried it on the testing website that I created. The page was composed of a single element with the class name "toblue", and a script that changed the "toblue" class to a "blue" class (which was still defined to make the text blue). This was the bug I already experienced with the getElementsByTagName method: changing the class of an element that already got one didn't seem to work. I had to figure out why and correct it.

The second one was in the definition of getElementsByClassName. I thought it could only accept a single class, but in fact, it's supposed to be able to accept several classes, separated by a space. For example, document.getElementsByClassName("a b") is supposed to return all elements that have both classes "a" and "b". So I had to take this into account.

6. End of the implementation

a. Changing an existing class

As stated previously, our first guess about the class not being changed was that the CSS interpreter stopped trying to handle changes. We supposed that by trying to log the class of an element before and after changing its class. The results were the ones expected from a logical point of vue (but not from what we were seeing on the screen) : the old class was “toblue”, but was changed to “blue” after the javascript execution. Since the alteration occurred, it seemed logical to think that the CSS interpreter was causing the problem. However, this was not really the case.

As I figured out, there was a cache to keep the CSS classes in memory. There were even methods to clear it… but they weren’t used, even in the method that was called whenever an attribute was changed. So, I modified this method to clear the cache if the attribute changed was a class. After that, both getElementsByTagName and getElementsByClassName successfully changed the class as they were supposed to.

b. Handling several classes

It was then time to finish the implementation of getElementsByClassName, by allowing a user to search for several classes at once. To do that, I had to split the input string into several others, one for each present class. The different classes are separated by an empty space in the input string. My first try was to use the C function strtok, which can be used to split a string into substrings given a specific delimiter. However, this led to two problems.

To understand the first problem, one need to understand the structures used by libdom. It is using structures that are not native in C - like char* (which represents a string). Instead, to represent a string of characters, it is using domstrings. To use a function of libdom, DukTape has to convert a char* variable to a domstring one. However, the opposite, though possible, is a read only output, because it returns a constant value. It can’t be altered without getting memory-related errors. One solution was to copy the content of this “const char*” into a new char*, but this didn’t solve the second problem.

The second issue is directly related to strtok. In fact, it is keeping the used string in memory to get the substrings. This means that it can’t be used in parallel programs, since it would be erased by the next call. So, if a webpage contained several getElementsByClassName, for example, this method wouldn’t work properly. Moreover, I already noticed that using char* was not working as expected, probably because of AmigaOS memory management.

To manage several classes, I then browsed the available functions that were implemented for domstrings. I found two that could allow me to split my strings. The first one was dom_string_substr, that, given a domstring and the 2 indexes for the beginning and the end of the string, creates a new domstring. The second one was dom_string_index, that, given a domstring and a character, returns the index of this character. With these two methods, I could easily split my initial string into substrings.

I then altered the “length” and “item” functions of the nodelist to take these changes into account. This made the getElementsByClassName fully functional !

V. Others

1. The unimplemented methods

During the development of the getElementsByClassName method, I had to browse the whole NetSurf project to have a global idea of it. I found a file, named UnimplementedJavascript.txt that contained … well, the unimplemented Javascript functions! There were a lot of them (1793 !!!). I quickly looked for some of them online, and found out that several were deprecated, and should no longer be used. So I began to search for all of these methods, to list the ones that don't need to be implemented in the future.

2. Trying to make alert and document.URL work

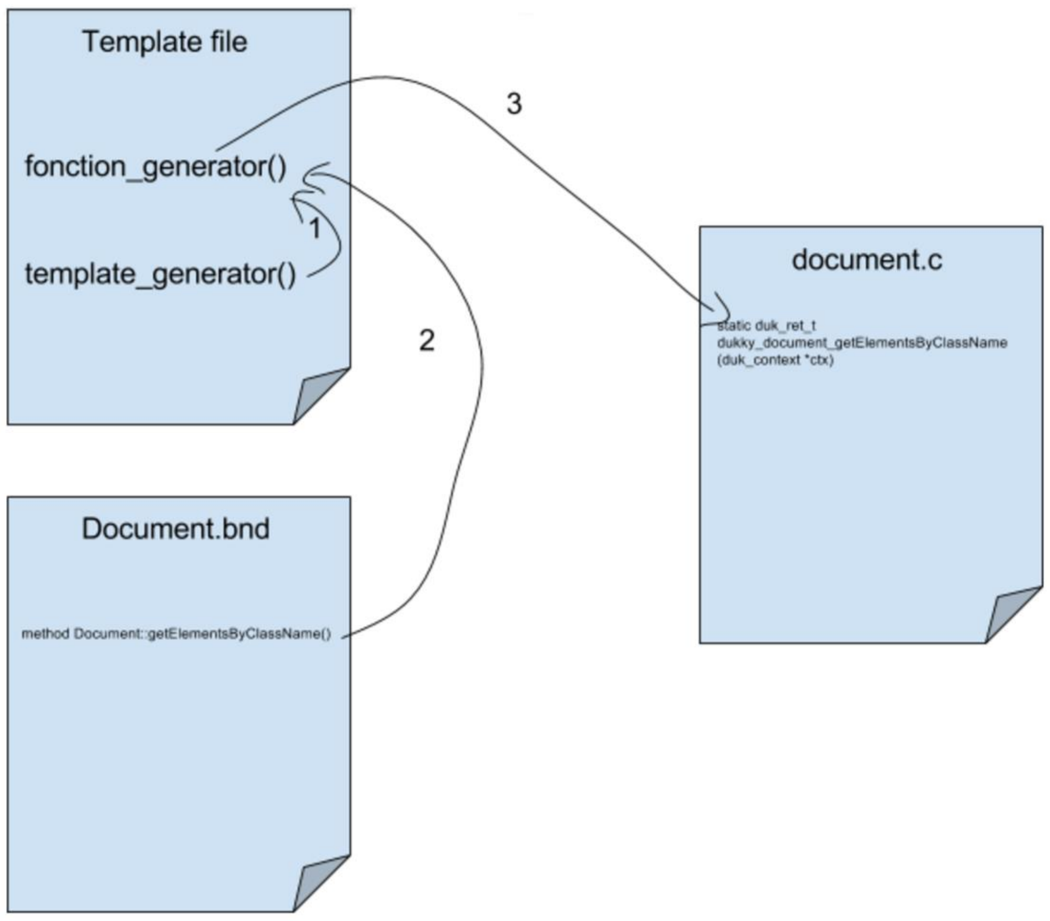
a. Alert

After implementing the getElementsByClassName method, I tried implementing a very popular JavaScript function: alert. This function pops an alert on the browser, with a button stating "OK", and some text in it. My first goal wasn't to make this small window pop, but to only log the content of the alert, with a message stating that it was called from the alert function, and not from the console.log method.

I didn't check the unimplemented methods file, because I already tried to use alert, and it didn't work. However, when I looked into the file where it was supposed to be implemented, I found out that it already contained an alert method! And when I checked, alert was indeed not in the unimplemented javascript file.

My first guess was that the problem came from the implementation. In fact, the alert function is supposed to have a very specific behaviour: it blocks the user from doing anything in the browser while the alert is present. However, when I analysed the code, it appeared that alert was implemented exactly as I wanted to in the first place: it was only logging the alert content. And there was no error in the code: it should have behaved exactly like the console.log method.

This part of the code is a bit tricky to understand. That is because the files containing this code (they're named X.bnd: for example, Document.bnd) aren't exactly the ones used after compilation. To be more precise, new files (x.c: for example, document.c) are made thanks to the bnd ones and some others that contain the code to create the new files and some templates. These "template" files are executed when compiling NetSurf. For each method, they first make the link between DukTape and libdom thanks to their templates, and then scan the bnd files to see whether an implementation exists in those. If one does, they copy it in the new file.



The issue is that libdom isn't completely finished. In particular, the templates aren't complete: some of them (basically, the one that checks the parameters types) are implemented, while others aren't. In libdom, they considered that the alert function is overloaded (it means this function can have different parameters: here a domstring, or no parameter). And the template for the overloaded functions hasn't been implemented yet. As this was very tricky to implement, I decided to not focus on the alert method.

b. Document.URL

After that, I tried to implement a getter: document.URL. It's supposed to get the URL of the current document. In fact, I thought this would be an easy thing to implement, since there is a method from html_document, that is supposed to do exactly what I wanted. However, after implementing document.URL and testing it, I realised that the method I was using always returned NULL. By looking into this method, I found it very simple: it returned the url property of the html_document structure. It means that this property is never filled in NetSurf. Even after searching for a while, I didn't find where it was supposed to be set, so I gave up this implementation.

3. Implementing other methods

In the file containing the unimplemented JavaScript methods, I also found some methods that worked identically for different objects, but were implemented in only one of them. A good example would be the childElementCount method. As its name suggests, it's a method returning the number of childs an element has. This function was implemented for the element class (so document.getElementsByTagName("p").item(0).childElementCount(), for example, was working), while it wasn't with the document class (so document.childElementCount(), was not working). So I quickly implemented these methods using the same functions used where they were implemented. I then tested them to be sure they were working as expected, by comparing their results to some "regular" browsers, like Google Chrome or Firefox.

VI. Project Management

Since the goal of the project was quickly changed, and since M. Guéhéneuc and me didn't know about the final output we were expecting, I couldn't make a Gantt diagram that wouldn't be completely guessed. The only things I put limits on were :

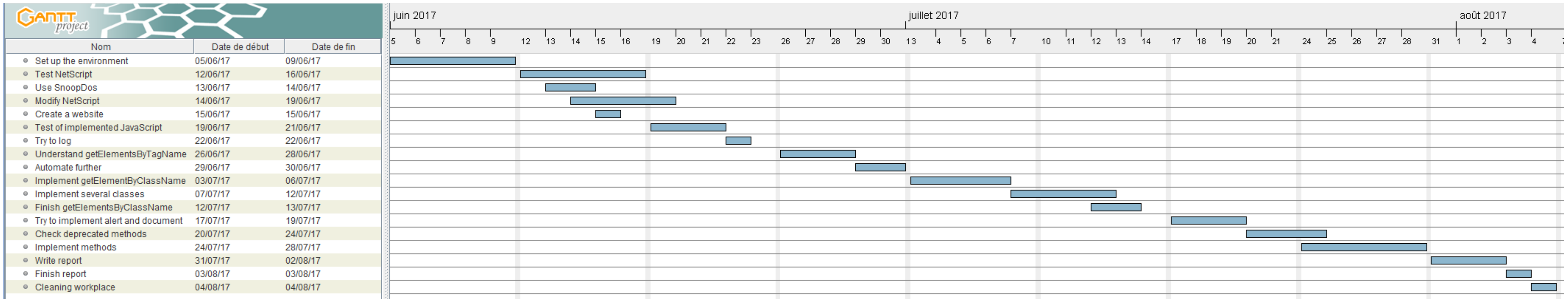
1. Focusing on the getElementsByClassName until completion
2. Giving up on other implementations if there was no progression within 3-5 days

On the next page, there is the Gantt diagram about how the things really went on.

All the tasks were described previously, so I'll focus on the holidays. The first and second ones are Quebec and Canada national days, so they are public holidays (they actually are on Saturday, but when it's the case, Canadians take Friday and sometimes Monday off). Then there is the PLOW event: conferences about Artificial Intelligence, the 4th and 5th of July. Finally, my tutor allowed me to take the other days off, so I could go to Toronto and New York.

About the communication, I tried to meet my tutor at least once a week to show him my progress. However, this was not possible all the time; when that was the case, I just wrote him a report by email instead.

I also made pull requests to the different open source projects (libdom library, NetSurf, NetScript, …) on GitHub, but I received no answer yet.



Nom	Date de début	Date de fin
Set up the environment	05/06/17	09/06/17
Test NetScript	12/06/17	16/06/17
Use SnoopDos	13/06/17	14/06/17
Modify NetScript	14/06/17	19/06/17
Create a website	15/06/17	15/06/17
Test of implemented JavaScript	19/06/17	21/06/17
Try to log	22/06/17	22/06/17
Understand getElementsByTagName	26/06/17	28/06/17
Automate further	29/06/17	30/06/17
Implement getElementByClassName	03/07/17	06/07/17
Implement several classes	07/07/17	12/07/17
Finish getElementsByClassName	12/07/17	13/07/17
Try to implement alert and document	17/07/17	19/07/17
Check deprecated methods	20/07/17	24/07/17
Implement methods	24/07/17	28/07/17
Write report	31/07/17	02/08/17
Finish report	03/08/17	03/08/17
Cleaning workplace	04/08/17	04/08/17

VII. Conclusion

This internship was a really good experience, both at the human and professional level. It allowed me to discover a new country, with other ways of thinking (both in everyday life and at work). I also had the opportunity to participate in a conference about Artificial Intelligence, and robot swamps programming, which described a whole new programming approach.

Moreover, I had the chance to practice English speaking and understanding. In fact, these 3 months were really interesting in this point of view, because Montreal's inhabitants tend to naturally switch between English and French, even in the middle of a sentence. This was completely new to me, and, in my opinion, not only does it improve English understanding, but also strengthen intellectual agility.

VIII. Glossary

Cygwin
Cygwin is a Shell emulator based on Unix/Linux systems for Windows. It allows to use commands such as "cd", "ls", … on a Windows computer.

NetSurf
NetSurf is a light web browser written in C/C++.

Duktape
DukTape is an embeddable JavaScript engine focused on portability. It is written in C/C++.

AmigaOS
AmigaOS is a family of operating systems of the Amiga and the AmigaOne personal computers. Its resources needs are very low.

Amiga Forever
Amiga Forever is an AmigaOS emulator.

Amikit
Amikit is a preconfigured package of more than 350 Amiga programs. It is available with Amiga Forever.

Libdom
Libdom is a library used by DukTape. Its goal is to implement the DOM requirements, for example the Javascript method getElementsByTagName.

NetScript
NetScript is a script which compiles NetSurf and its libraries to be used on AmigaOS.

IX. Web pages

AmigaOS - Wikipedia. (n.d.). https://en.wikipedia.org/wiki/AmigaOS

Amiga Forever. (n.d.). https://www.amigaforever.com/

Amikit. (n.d.). https://www.amikit.amiga.sk/

Cygwin - Wikipedia. (n.d.). https://en.wikipedia.org/wiki/Cygwin

document.getElementsByClassName method - mozilla (n.d.).

https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByClassName

DukTape (n.d.). http://duktape.org/

DukTape API (n.d.). http://duktape.org/api.html

Heroku (n.d.). https://www.heroku.com/

NetScript - GitHub. (n.d.). https://github.com/DNADNL/NetScript

NetSurf - GitHub. (n.d.). http://source.netsurf-browser.org/

NetSurf for AmigaOS3 - GitHub. (n.d.). https://github.com/EyMenZ/NetSurf-OS3

Ptidej team (n.d.). http://www.ptidej.net/

Strtok function - tutorialpoints (n.d.).

https://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm

X. Abstract

During this 3-month internship in Polytechnique Montréal, in Canada, I had to work on a light web browser, NetSurf, on a small OS, AmigaOS. The goal was to understand why this browser couldn't be launched when compiled with DukTape, a JavaScript engine. When this problem was solved, I then had to try and understand as much of DukTape, as well as some libraries like libdom, as I could. Finally, I implemented some methods to libdom and DukTape, so NetSurf would support more JavaScript thanks to DukTape.

Keywords : NetSurf, DukTape, JavaScript, AmigaOS, Montréal

Pendant ce stage de 3 mois à Polytechnique Montréal, au Canada, j'ai dû travailler sur un navigateur web léger, NetSurf, sur un système d'exploitation dont le principal atout est sa légèreté. L'objectif initial était de découvrir pourquoi le navigateur ne pouvait être lancé s'il était compilé avec DukTape, un interpréteur Javascript. Dès que ce problème fut résolu, j'ai pu me concentrer sur la compréhension globale de DukTape, ainsi que des librairies utilisées par ce dernier, tel libdom. Enfin, j'ai pu implémenter certaines méthodes à DukTape, libdom, … afin que NetSurf puisse supporter certaines fonctions en JavaScript.

Mots-clés : NetSurf, DukTape, JavaScript, AmigaOS, Montréal

XI. Appendices

1. NetScript.sh :

```
./NS.sh "$@" 2>&1 | tee LOG_NetScript.txt
```

2. NS.sh :

See the attached file

3. getElementsByClassName method :

```
method Document::getElementsByClassName()
%{
	struct dom_nodelist *nodes;
	dom_exception err;
	duk_size_t text_len;
	//TIP : La classe a rechercher : en ctx 0
	const char *text = duk_safe_to_lstring(ctx, 0, &text_len);
	dom_string *clas;

	err = dom_string_create((uint8_t*)text, text_len, &clas);

	if (err != DOM_NO_ERR) return 0; /* coerced to undefined */

	LOG("La classe a rechercher est %s de taille %d", text, text_len);

	err = dom_document_get_elements_by_class_name(((node_private_t
*)priv)->node, clas, &nodes);

	LOG("err vaut %d", err);

	dom_string_unref(clas);

	uint32_t *len;
	dom_nodelist_get_length(nodes, len);
	LOG("On trouve : %d element(s)", (*len));
	LOG("debug : '%s'", dom_string_data(debug));
	LOG("debug2 : '%s'", dom_string_data(debug2));

	if (err != DOM_NO_ERR) return 0; /* coerced to undefined */

	if (nodes == NULL) return 0; /* coerced to undefined */
	duk_push_pointer(ctx, nodes);
	dukky_create_object(ctx, PROTO_NAME(NODELIST), 1);
	dom_nodelist_unref(nodes);

	return 1;
%}
```

4. The new nodelist structure :

```
struct dom_nodelist {
	dom_document *owner;	/**< Owning document */

	dom_node_internal *root;
			/**< Root of applicable subtree */

	nodelist_type type;	/**< Type of this list */

	union {
		struct {
			dom_string *name;
					/**< Tag name to match */
			bool any_name;		/**< The name is '*' */
		} n;
		struct {
			dom_string *classname;
					/**< Class name to match */
		} cn;
		struct {
			bool any_namespace;	/**< The namespace is '*' */
			bool any_localname;	/**< The localname is '*' */
			dom_string *namespace;	/**< Namespace */
			dom_string *localname;	/**< Localname */
		} ns;			/**< Data for namespace matching */
	} data;

	uint32_t refcnt;		/**< Reference count */
};
```

5. The _dom_document_get_elements_by_class_name function :

```
dom_exception _dom_document_get_elements_by_class_name(dom_document *doc,
		dom_string *classname, dom_nodelist **result)
{
	return _dom_document_get_nodelist(doc, DOM_NODELIST_BY_CLASSNAME,
(dom_node_internal *) doc,  NULL, classname, NULL, NULL, result);
}
```

6. The new dom_document_get_nodelist function :

```
dom_exception _dom_document_get_nodelist(dom_document *doc,
            nodelist_type type, dom_node_internal *root,
            dom_string *tagname, dom_string *classname, dom_string
*namespace,
            dom_string *localname, dom_nodelist **list)
{
      struct dom_doc_nl *l;
      dom_exception err;

      for (l = doc->nodelists; l; l = l->next) {
            if (_dom_nodelist_match(l->list, type, root, tagname,
classname, namespace, localname))
                  break;
      }

      if (l != NULL) {
            /* Found an existing list, so use it */
            dom_nodelist_ref(l->list);
      } else {
            /* No existing list */

            /* Create active list entry */
            l = malloc(sizeof(struct dom_doc_nl));
            if (l == NULL)
                  return DOM_NO_MEM_ERR;

            /* Create nodelist */
            err = _dom_nodelist_create(doc, type, root, tagname,
classname, namespace, localname, &l->list);

            if (err != DOM_NO_ERR) {
                  free(l);
                  return err;
            }

            /* Add to document's list of active nodelists */
            l->prev = NULL;
            l->next = doc->nodelists;
            if (doc->nodelists)
                  doc->nodelists->prev = l;
            doc->nodelists = l;
      }

      /* Note: the document does not claim a reference on the nodelist
      * If it did, the nodelist's reference count would never reach zero,
      * and the list would remain indefinitely. This is not a problem as
      * the list notifies the document of its destruction via
      * _dom_document_remove_nodelist. */

      *list = l->list;
```

```
	return DOM_NO_ERR;
}
```

7. The new item function from a nodelist :

```
dom_exception _dom_nodelist_item(dom_nodelist *list,
		uint32_t index, dom_node **node)
{
	dom_node_internal *cur = list->root->first_child;
	uint32_t count = 0;

	/* Traverse data structure */
	while (cur != NULL) {
		/* Process current node */
		if (list->type == DOM_NODELIST_CHILDREN) {
			count++;
		} else if (list->type == DOM_NODELIST_BY_NAME) {
			if (list->data.n.any_name == true || (
					cur->name != NULL &&
					dom_string_isequal(cur->name,
						list->data.n.name))) {
				if (cur->type == DOM_ELEMENT_NODE)
					count++;
			}
		} else if (list->type == DOM_NODELIST_BY_NAME_CASELESS) {
			if (list->data.n.any_name == true || (
					cur->name != NULL &&
					dom_string_caseless_isequal(cur->name,
						list->data.n.name))) {
				if (cur->type == DOM_ELEMENT_NODE)
					count++;
			}
		} else if (list->type == DOM_NODELIST_BY_CLASSNAME) {
			if (cur->type == DOM_ELEMENT_NODE){
				bool hasClass = false;
				bool hasAllClasses = true;

				lwc_string *str = NULL;
				char *temp = NULL;
				dom_string *temp_str;
				dom_string_copy(list->data.cn.classname,
&temp_str);
				dom_string *temp_class = NULL;
				uint32_t end = dom_string_index(temp_str, ' ');
				uint32_t leng = dom_string_length(temp_str);

				while(end != (uint32_t)-1){
					if (end == 0){
						dom_string_substr(temp_str, 1, leng,
&temp_str);
						leng = dom_string_length(temp_str);
```

```
                                    end = dom_string_index(temp_str, ' ');
                              } else if(end == (leng-1)) {
                                    dom_string_substr(temp_str, 0, leng-1,
&temp_str);

                                    leng = dom_string_length(temp_str);
                                    end = dom_string_index(temp_str, ' ');
                              } else {
                                    dom_string_substr(temp_str, 0, end,
&temp_class);

                                    dom_string_substr(temp_str, end+1,
leng, &temp_str);

                                    leng = dom_string_length(temp_str);
                                    end = dom_string_index(temp_str, ' ');

                                    temp = (char
*)dom_string_data(temp_class);

                                    lwc_intern_string(temp, strlen(temp),
&str);

                                    _dom_element_has_class((dom_element
*)cur, str, &hasClass);

                                    if(!hasClass)
                                          hasAllClasses = false;
                              }
                        }

                        if (leng > 0){
                              temp = (char *)dom_string_data(temp_str);

                              lwc_intern_string(temp, strlen(temp), &str);

                              _dom_element_has_class((dom_element *)cur,
str, &hasClass);

                              if(!hasClass)
                                    hasAllClasses = false;
                        }

                        lwc_string_unref(str);
                        dom_string_unref(temp_str);
                        dom_string_unref(temp_class);
                        if (hasAllClasses)
                              count++;
                  }
            } else if (list->type == DOM_NODELIST_BY_NAMESPACE) {
                  if (list->data.ns.any_namespace == true ||
                              (cur->namespace != NULL &&
                              dom_string_isequal(cur->namespace,
                                    list->data.ns.namespace))) {
                        if (list->data.ns.any_localname == true ||
                                    (cur->name != NULL &&
                                    dom_string_isequal(cur->name,
                                list->data.ns.localname))) {
                        if (cur->type == DOM_ELEMENT_NODE)
                                count++;
                }
        }
} else if (list->type == DOM_NODELIST_BY_NAMESPACE_CASELESS)
{
        if (list->data.ns.any_namespace == true ||
                        (cur->namespace != NULL &&
                        dom_string_caseless_isequal(
                                cur->namespace,
                                list->data.ns.namespace))) {
                if (list->data.ns.any_localname == true ||
                                (cur->name != NULL &&
                                dom_string_caseless_isequal(
                                cur->name,
                                list->data.ns.localname))) {
                        if (cur->type == DOM_ELEMENT_NODE)
                                count++;
                }
        }
} else {
        assert("Unknown list type" == NULL);
}

/* Stop if this is the requested index */
if ((index + 1) == count) {
        break;
}

/* Now, find next node */
if (list->type == DOM_NODELIST_CHILDREN) {
        /* Just interested in sibling list */
        cur = cur->next;
} else {
        /* Want a full in-order tree traversal */
        if (cur->first_child != NULL) {
                /* Has children */
                cur = cur->first_child;
        } else if (cur->next != NULL) {
                /* No children, but has siblings */
                cur = cur->next;
        } else {
                /* No children or siblings.
                * Find first unvisited relation. */
                dom_node_internal *parent = cur->parent;

                while (parent != list->root &&
                                cur == parent->last_child) {
                        cur = parent;
                        parent = parent->parent;
                }
```

```
                        cur = cur->next;
                  }
            }
      }

      if (cur != NULL) {
            dom_node_ref(cur);
      }
      *node = (dom_node *) cur;

      return DOM_NO_ERR;
}
```

8. The new length function from a nodelist :

```
dom_exception dom_nodelist_get_length(dom_nodelist *list, uint32_t
*length)
{
      dom_node_internal *cur = list->root->first_child;
      uint32_t len = 0;
      /* Traverse data structure */
      while (cur != NULL) {
            /* Process current node */
            if (list->type == DOM_NODELIST_CHILDREN) {
                  len++;
            } else if (list->type == DOM_NODELIST_BY_NAME) {
                  if (list->data.n.any_name == true || (
                              cur->name != NULL &&
                              dom_string_isequal(cur->name,
                                    list->data.n.name))) {
                        if (cur->type == DOM_ELEMENT_NODE)
                              len++;
                  }
            } else if (list->type == DOM_NODELIST_BY_NAME_CASELESS) {
                  if (list->data.n.any_name == true || (
                              cur->name != NULL &&
                              dom_string_caseless_isequal(cur->name,
                                    list->data.n.name))) {
                        if (cur->type == DOM_ELEMENT_NODE)
                              len++;
                  }
            } else if (list->type == DOM_NODELIST_BY_CLASSNAME) {
                  if (cur->type == DOM_ELEMENT_NODE){
                        bool hasClass = false;
                        bool hasAllClasses = true;

                        lwc_string *str = NULL;
                        char *temp = NULL;
                        dom_string *temp_str;
                        dom_string_copy(list->data.cn.classname,
&temp_str);
                        dom_string *temp_class = NULL;
                        uint32_t end = dom_string_index(temp_str, ' ');
                        uint32_t leng = dom_string_length(temp_str);

                        while(end != (uint32_t)-1){
                              if (end == 0){
                                    dom_string_substr(temp_str, 1, leng,
&temp_str);
                                    leng = dom_string_length(temp_str);
                                    end = dom_string_index(temp_str, ' ');
                              } else if(end == (leng-1)) {
                                    dom_string_substr(temp_str, 0, leng-1,
&temp_str);
                                    leng = dom_string_length(temp_str);
                                    end = dom_string_index(temp_str, ' ');
                              } else {
                                    dom_string_substr(temp_str, 0, end,
&temp_class);
                                    dom_string_substr(temp_str, end+1,
leng, &temp_str);
                                    leng = dom_string_length(temp_str);
                                    end = dom_string_index(temp_str, ' ');

                                    temp = (char
*)dom_string_data(temp_class);

                                    lwc_intern_string(temp, strlen(temp),
&str);

                                    _dom_element_has_class((dom_element
*)cur, str, &hasClass);
                                    if(!hasClass)
                                          hasAllClasses = false;
                              }
                        }

                        if (leng > 0){
                              temp = (char *)dom_string_data(temp_str);

                              lwc_intern_string(temp, strlen(temp), &str);

                              _dom_element_has_class((dom_element *)cur,
str, &hasClass);
                              if(!hasClass)
                                    hasAllClasses = false;
                        }

                        lwc_string_unref(str);
                        dom_string_unref(temp_str);
                        dom_string_unref(temp_class);
                        if (hasAllClasses)
```

```
                              len++;
                  }
            } else if (list->type == DOM_NODELIST_BY_NAMESPACE) {
                  if (list->data.ns.any_namespace == true ||
                              dom_string_isequal(cur->namespace,
                              list->data.ns.namespace)) {
                        if (list->data.ns.any_localname == true ||
                                    (cur->name != NULL &&
                                    dom_string_isequal(cur->name,
                                    list->data.ns.localname))) {
                              if (cur->type == DOM_ELEMENT_NODE)
                                    len++;
                        }
                  }
            } else if (list->type == DOM_NODELIST_BY_NAMESPACE_CASELESS)
{
                  if (list->data.ns.any_namespace == true ||
                              dom_string_caseless_isequal(
                              cur->namespace,
                              list->data.ns.namespace)) {
                        if (list->data.ns.any_localname == true ||
                                    (cur->name != NULL &&
                                    dom_string_caseless_isequal(
                                    cur->name,
                                    list->data.ns.localname))) {
                              if (cur->type == DOM_ELEMENT_NODE)
                                    len++;
                        }
                  }
            } else {
                  assert("Unknown list type" == NULL);
            }

            /* Now, find next node */
            if (list->type == DOM_NODELIST_CHILDREN) {
                  /* Just interested in sibling list */
                  cur = cur->next;
            } else {
                  /* Want a full in-order tree traversal */
                  if (cur->first_child != NULL) {
                        /* Has children */
                        cur = cur->first_child;
                  } else if (cur->next != NULL) {
                        /* No children, but has siblings */
                        cur = cur->next;
                  } else {
                        /* No children or siblings.
                        * Find first unvisited relation. */
                        dom_node_internal *parent = cur->parent;

                        while (parent != list->root &&
                                    cur == parent->last_child) {
```

```
                                                cur = parent;
                                                parent = parent->parent;
                                        }

                                        cur = cur->next;
                                }
                        }
                }

                *length = len;

                return DOM_NO_ERR;
        }
```

9. The end of the script launched at AmigaOS start :

```
cd NetScript-master:
AmiKit:System/CLI
```

10. The script that launches NetSurf in verbose mode with an output file :

```
NetScript-master:NetSurf_3.6dev_WithDukTape_AmigaOS3/NetSurf/NetSurf -v
*> NetScript-master:NetSurf_3.6dev_WithDukTape_AmigaOS3/NetSurf/test.txt
```

11. The default settings in NetSurf :

```
font_size:120
font_min_size:80
enable_javascript:1
homepage_url:http://testjscript.herokuapp.com/
```