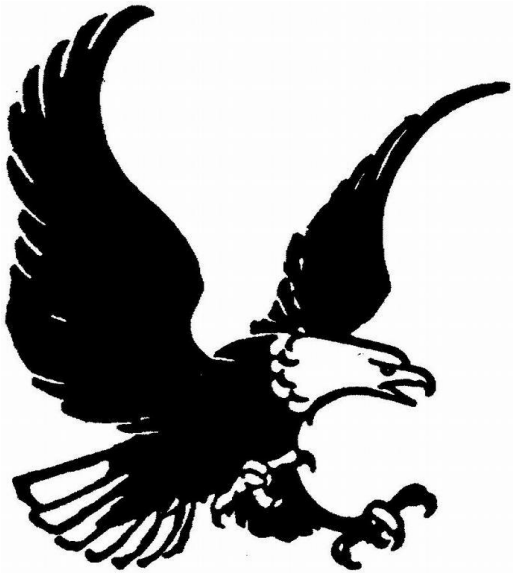


CAEWS

CONTEMPORARY AND EASY WEB SCRIPT

IDA ENBRANT

Innovativ Programmering VT 2012
TDP019 – Projekt: Datorspråk
Linköpings universitet

SAMMANFATTNING

Caews är ett enkelt web script, tänkt att användas för att bygga privata hemsidor utan att behöva kunna avancerad webprogrammering. Språket har en enkel syntax som är lätt att lära sig. Caews har funktioner som gör det möjligt att lägga all kod i samma dokument, och har vissa element som gör att det påminner mer om ett programmeringsspråk än ett web script.

Caews är byggt i Ruby 1.9, och kräver att Rubys kompilator finns installerad för att köras. Språket genererar html och css, som fungerar i de flesta webbläsare. Det mesta av koden översätts till bägge språken. Som html har Caews taggar som omsluter koden, men till skillnad från detsamma behöver Caews inte alltid sluttaggar.

Caews använder BNF-grammatik precis som många programmeringsspråk. Grammatiken är förhållandevis simpel för att göra det enkelt att översätta i parsern som bearbetar koden och sköter översättningen.

Projektet Caews har varit tidskrävande men väldigt lärorikt och förväntas vara mycket användbart i framtiden.

INNEHÅLLSFÖRTECKNING

1 Inledning

Caews – Contemporary And Easy Web Script – är frukten av ett projekt som startades upp under våren 2012 av författaren, Ida Enbrant. I dagens samhälle, där många tjänster domineras av Internet och nästan varje hem har en dator, så kändes det rätt i tiden att låta projektet röra ett web script[1]. Caews blev resultatet – ett modernt och lättfattligt datorspråk för att snabbt designa hemsidor. En inkörsport för den ovane ”hemsidesnickraren”, men också ett praktiskt verktyg för den vana webbutvecklaren då Caews kräver mindre tid i anspråk.

Syftet med Caews är just det – att vara en genväg, med mindre kod för större arbeten, som effektiviserar skapandet av moderna hemsidor. Den enkla, lätta koden är tänkt att låta användaren lätt och snabbt bygga hemsidor som i dagsläget tar både mycket kunskap och tid i anspråk. Som fokus har Caews de typer av hemsidor och designval som är vanligast förekommande idag, för att språket ska kännas aktuellt i tiden.

Det som gör Caews speciellt, är användandet av så kallade *virtuella fönster*. De virtuella fönstren fungerar som egna objekt utan att skrivas i separata filer, och gör Caews till ett väldigt flexibelt och snabbarbetat språk. Caews har också, likt många programmeringsspråk, möjligheten att lagra kod i variabler[2] som användaren själv kan döpa.

Dokumentationen består av en användarhandledning som är avsedd att vara lätt att förstå, för att göra det smidigt att komma igång med användandet av Caews.

[1] Ett *web script* är ett domänspecifikt programmeringsspråk som används för att skapa hemsidor.
[2] *Variabler* är namn på stycken av kod.

2 Dokumentation

Dokumentationen är tänkt att vara en lättförståelig manual till att använda Caews. Då fokus ligger i design och komposition, är dokumentationen uppdelad i de olika designstegen med konkreta exempel på hur koden kan se ut. Caews är flexibelt i den mån att de flesta attribut[3] delas av fler än en tag[4], så användaren uppmuntras att experimentera med attributen.

2.1 Installera Caews

Caews går att använda på de system och webbläsare som stöder HTML, CSS samt Javascript[5]. Dessa är vanligt förekommande i moderna system. Värt att komma ihåg är att resultatet kan se något annorlunda ut i olika webbläsare beroende på språkstödet. Caews behöver inte något ramverk[6] – koden genererar språk som de flesta webbläsare förstår.

Caews är utvecklat i programmeringsspråket Ruby[7], vilket kräver att Ruby finns installerat med fullt språkstöd. För att installera Ruby, besök följande adress: **http://www.ruby-lang.org/en/**, och klicka på *Download Ruby*. Det finns flera olika operativsystem och flera olika sätt att installera Ruby. Välj det sätt som passar dig och ditt operativsystem bäst.

För att använda Caews behöver du kopiera följande filer till den mapp du har ditt projekt i:

cws_parser.rb
cws_run.rb

För att skriva kod i Caews går det alldeles utmärkt att använda valfri texteditor[8], koden sparas sedan med ändelsen '.cws'. Exempel: min_kod.cws. Spara filen i samma mapp som du vill ha din hemsida.

2.2 Kompilering

För att din kod ska bli användbar, och din hemsida faktiskt skapas, måste koden du skrivit *kompileras*. Det innebär att koden måste köras, och Caews omvandla din kod till något din dator förstår. När Caews kompileras, skapas filer i samma mapp som du sparat din Caews-fil i. Du kommer att behöva samtliga av dessa filer för att din hemsida ska fungera korrekt. Caews har i dagsläget inte ett grafiskt läge som gör att du hela tiden kan se dina uppdateringar, utan kräver att koden kompileras efter ändringar för att resultatet ska kunna kontrolleras.

Caews kompileras i Ruby, så vid kompilering är det viktigt att starta upp en Ruby-interpretator eller en terminal i Ruby-läge (också kallad 'irb'). För att Ruby ska hitta filen måste du först antingen öppna filen *cws_run.rb* genom att dubbelklicka på den.

---

[3] *Attribut* i Caews är de egenskaper som gör mindre justeringar av kod.
[4] *Tag* är ett Engelskt ord som betyder *etikett*, och i detta sammanhang används om kod som befinner sig inom symbolerna '<' och '>'. *Slut-tag* hänvisar till en tag inom symbolerna '</' och '>'.
[5] HTML, CSS samt Javascript är webscript som är vanliga vid webbutveckling.
[6] Ett *ramverk* är ett program som kan tolka en viss typ av kod.
[7] *Ruby* är ett objektorienterat programspråk utvecklat av Yukihiro Matsumoto. Caews använder 1.9.3-versionen av Ruby.
[8] En *texteditor* är ett program som inte formaterar texten, exempelvis Anteckningar/Notepad. Det finns många olika texteditors som är användbara, vissa mer kodorienterade än andra.

Du får nu upp ett litet program som ber dig att skriva in namnet på din fil. Kom ihåg att ändelsen måste vara med!

Det är allt som krävs. Nu gör Caews allt jobb – tolkar din kod samt skapar två nya dokument i samma mapp som din fil. Dessa två dokument kommer att heta samma sak som din fil men ha filändelserna .css och .html. Det är dessa två filer som du kommer att behöva till din hemsida, i kombination med eventuella bilder. HTML-filen är den fil där ditt arbete visas.

2.3 Komma igång

Innan du börjar skriva din Caews-kod kan det vara en god idé att tänka över vad det är du vill göra. Hur vill du att din hemsida ska se ut, och hur ska du be Caews göra detta åt dig? Caews är inriktat på modern hemsidedesign, exempelvis för en blogg[9] eller en hemsida med fönsterbaserad layout. Det kan vara en god idé att använda exemplen som följer för att bygga en bas, som du sedan kan justera efter tycke och smak. Notera att alla namn på bilder och filer är exempel, och givetvis inte finns på just din dator om du inte medvetet namnger dina bilder och filer efter exemplen. All kod går att skriva i en och samma Caews-fil.

Caews har väldigt många attribut som delas av många tags, vilket innebär att för att undvika upprepningar finns varje attribut beskrivet på ett begränsat antal platser. Attributen fungerar dock – om ingenting annat sägs – på samma sätt för alla tags som delar dem.

2.4 Bakgrunder

Det finns två sätt att göra bakgrunder i Caews: enfärgade bakgrunder eller använda en bild som bakgrund. Båda sätten använder tagen *background*, som i sin enklaste form ser ut som följer:

```
<background>
```

Den enkla grund-tagen utan attribut ger en vit, ogenomskinlig bakgrund. För att justera *background* finns en mängd olika attribut som kan läggas till. Attributet som justerar färg heter *colour*. Så här ser *background* ut om attributet *colour* läggs till och ges ett värde:

```
<background colour=blue>
```

Tagen ger dig nu en helt blå bakgrund. De grundläggande, vanligaste färgerna har nyckelord som kan skrivas med bokstäver, men det är möjligt – och ofta önskvärt – att använda *Hexadecimaler*[10]. En enkel färgkarta över Hexadecimala färger finns i bilagorna.

För att använda en bild som bakgrund hellre än en färg, används attributet *image*. *Image* kräver adressen till en bild – lokalt med bildens mappnamn som adress, eller en URL[11]. Till skillnad från många språk behövs inte några citationstecken runt adressen. Notera att det inte får finnas några mellanslag i adressen.

---

[9] En *blogg* är en online-journal, där användare kan kommentera varandras dagboksinlägg.
[10] Hexadecimala är ett numeriskt system som gör om siffror och bokstäver till en kod, och som används till exempelvis färgkodning. Till färger kan hexadecimal exempelvis se ut som följande för färgen vit: #FFFFFF.
[11] En *URL* är en hemsideadress/bildadress till en online källa.

<background image=/bilder/min_bakgrundsbild.png>

Det går att lägga till obegränsat antal attribut inom *background*s tag – det går även att ha fler attribut som gör liknande saker utan att de krockar med varandra. Om både *colour* och *image* finns med, väljer Caews automatiskt att visa *image*. Om bilden inte hittas, visas istället *colour*. Skrivs hela *background*s tag på fler ställen i samma fil, så används den senaste definierade variationen.

Följande attribut finns att tillgå i tagen *background:* transparency, width, height, fixed, stretch, name.

2.5 Typsnitt och text

Det går givetvis att skriva vanlig text i Caews, som skrivs ut oformaterad med standardtypsnitt, färg och storlek, men det blir något mer intressant om texten anpassas. Tagen som används för att formatera text heter *text*, och ser ut som följer:

<text>

Den enkla tagen utan attribut ger svart text med standardtypsnittet Arial, storlek 10px[12]. För att själv justera dessa standardvärden används attributen *font* och *size*. *Font* kräver ett typsnitt som din dator har tillgång till, så det kan vara en god idé att välja ett typsnitt som är vanligt. Anledningen är att de som tittar på din hemsida kanske inte har tillgång till samma utbud av typsnitt, och hemsidan kan se annorlunda ut.

<text font=Calibri size=12>

Om typsnittet har mellanslag, som exempelvis *Times New Roman*, måste citationstecken sättas om ordet, så att Caews förstår att alla orden hör ihop. Storleken anges endast med siffror, bokstäverna *px* behöver inte skrivas ut. Det går att använda de attribut du önskar ändra, övriga kan utelämnas. De attribut som inte anges antar förbestämda standardvärden.

<text size=20>

Caews kommer nu att utgå från sitt standardtypsnitt men ge texten storlek 20. *Text* har också något som kallas *slut-tag*, om du vill sluta använda textformatteringen du angett:

<end text>

Efter *end text* återgår Caews till standardformatteringen. Finns fler *text*-tags i koden, går den tillbaka till den formattering som användes senast.

Följande attribut finns att tillgå i tagen *text:* name.

2.6 Bilder

Bilder i Caews läggs till med tagen *image*, som fungerar på samma sätt som attributet *image*, med undantaget för att den har en egen tag:

<image>

[12] Px står för *pixlar*, och anger höjden på bokstäverna.

Image i sig själv gör inte någonting; den kräver en adress till en bild för att fungera. För att lägga en bildadress till *image* används samma attribut som till bakgrunder: *image*. Anledningen till varför *image* återanvänds är för att göra det lätt att komma ihåg, samt för att skilja den från attributet *adress,* som vi kommer gå igenom i avsnittet om länkar. Alla bildadresser oavsett huvud-tag använder attributet *image* för bildadresser.

<image image=/bilder/min_bild.jpg>

Följande attribut finns att tillgå i tagen *image:* transparency, width, height, name.

2.7 Repetition

Till skillnad från de flesta skript – och i likhet med många programmeringsspråk – har Caews *loopar*[13]. Loopar i det här fallet refererar till en viss bit kod som upprepas, och heter *switch*. *Switch* har en öppnings-tag och en slut-tag, och upprepar varje tag som finns mellan dessa. Den öppnas upp med följande:

<switch>

Switch är i sig inte oändlig, utan kör alla tags en gång tills den når sin slut-tag:

<end switch>

För att *switch* ska fortsätta även efter sin slut-tag måste attributet *repeat* användas. *Repeat* är en *boolean*[14], med värde *yes* eller *no*.

<switch repeat=yes>

Om ingenting annat anges, visar den varje bit av kod i 5 sekunder innan den går vidare till nästa tag. För att bestämma hur länge koden ska visas, används attributet *sleep*. *Sleep* kräver en siffra, och siffran i sig representerar en sekund.

<switch sleep=10>

För att göra övergången mellan tagsen bättre, finns attributed *fade*, som automatiskt är aktiverad när switch används. *Fade* gör helt enkelt att koden bleknar och går över i nästa. *Fade* är, precis som *repeat*, en boolean med värde *yes* eller *no*. Vid *no* inaktiveras fade.

<switch sleep=10 fade=no>

Switch är väldigt användbar till exempelvis bildspel, där tagsen mellan switch-tagarna helt enkelt använder tagen *image*. Detta kan se ut som följande:

<switch sleep=6 fade=yes repeat=yes>

<image image=/bilder/min_bild01.jpg>

<image image=/bilder/min_bild02.jpg>

[13] En *loop* är en sekvens av kod som upprepas angett antal gånger. Det finns givetvis evighets-loopar.
[14] *Boolean* är sanningsvärden, och kan vara antingen Sant eller Falskt.

<end switch>

2.8 Länkar

Länkar i Caews kan göras på två sätt: textlänkar eller bildlänkar. Båda använder tagen *link*. *Link* ser ut följer:

<link>

Om ingenting anges, är *link* tom och gör ingenting. För att länken ska leda någonstans, används attributet *adress* för att ange den URL eller det *fönster* som link ska peka på. URLen kan antingen vara *lokal* eller leda till en annan adress på internet. För att kunna göra en länk till en lokal sida behöver namnet på filen vara känt. Att länka till *fönster* är en av Caews speciella egenskaper, så detta kommer i nästa avsnitt.

<link adress=http://my_page.com/>

Link har attributet *image* precis som tidigare, vilket används för att göra bildlänkar. När *image* anges blir bilden en länk som går att klicka på. För att istället göra en *textlänk* används attributet *text* följt av text inom citationstecken.

<link adress=http://my_page.com/ text=”min textlänk”>

Det är viktigt att antingen ange text eller bild när en länk skapas, annars är den osynlig.

Följande attribut finns att tillgå i tagen *link:* type, colour, border, name.

2.9 Attribut

Det finns många attribut i Caews, och många av dem delas av flera tags. Attributen i sig har beskrivande namn, men det är också viktigt att attributen får ett *värde*[15] som Caews förstår. Det finns två standardtyper av värden, samt tre speciella typer. Standardtyperna kräver antingen heltal eller strängar[16]. Exempel på attribut som använder heltal är följande:

Size, width, height, x_position, y_position, sleep

Exempel på attribut som väntar sig strängar är följande:

Text, adress, image, font

De tre speciella typerna har väldigt specifika förväntade värden. Först och främst är givetvis *booleans*, som kräver antingen *yes* eller *no* (alternativt *true* och *false*) som värde. Booleans kan inte ta något annat värde. Exempel på booleanska attribut är följande:

Fade, scroll, repeat

[15] *Värde* (Eng. value) används ofta om det som står till vänster om tilldelningsoperatorn ’=’.
[16] Eng. string. Hänvisar till en sammanhängande sträng av bokstäver alt. symboler eller siffror.

Den andra typen av speciella attribut är de som kräver värden som är *nyckelord*. Nyckelorden är ett antal förutbestämda värden som Caews förväntar sig. Exempel på dessa är följande:

Font, colour

Till sist finns det speciella attributet *name*, som har en helt egen kategori: *variabler*. Variabler används frekvent inom programmeringsspråk, men är mindre vanligt förekommande i skriptspråk. Vad attributet *name* gör, är att det tillåter användaren att spara en tag med ett eget namn. Det får inte förekomma mellanslag i variabler. Exempelvis:

<background colour=pink name=my_background>

Nu har vi skapat en bakgrund med namnet *my_background*, som vi nu kan återanvända någon annanstans genom att hantera *my_background* som en ny tag:

<my_background>

My_background antar extakt samma värden som tagen vi gav namnet, och det är då inte nödvändigt att upprepa kod. I övrigt hanteras den nya tagen precis som vi skulle hanterat *background* – det går att lägga till attribut med nya värden inom *my_background*s tags.

2.10 Ramar och fönster

En av Caews speciella tags är *window*. *Window* är ett fönster – en virtuell del av hemsidan som har helt egen kod. För att kunna använda *window* krävs det att du först gör utrymme för dem, med hjälp av special-tagen *make_frame*. *Make_frame* används endast till att göra plats åt fönster, och dess attribut är viktiga. Det viktigaste att komma ihåg för att använda make_frame är att den måste ha attributet *name* – en variabel – för att den nya ramen för fönstret ska kunna användas.

För att skapa ramen krävs följande tag:

<make_frame name=frame_name>

Variabeln *frame_name* kan heta vad som helst – det enda kravet är att det är en sträng utan mellanslag, och utan vissa speciella symboler[17] – men låt oss använda *frame_name* som exempel. En ram har, precis som de flesta tags, grundvärden om ingenting annat anges. De viktigaste attributen för en ram är dess storlek, som anges med attributen *width* och *height*. Dessa två kräver heltal som anger storleken i antalet pixlar.

<make_frame name=frame_name width=300 height=300>

Nu när ramen är skapad och är 300x300 pixlar stor, är det lämpligt att tala om för Caews var på hemsidan ramen ska ligga. Det görs genom att ange x- och y-koordinater[18] med hjälp av attributen *x_position* samt *y_position*. Origo i grafen befinner sig högst upp i vänstra hörnet, så det är den punkt Caews utgår från när värden anges.

[17] Vanliga symboler som bör undvikas är <, >, ”, ’, ., (,) och * .
[18] X och Y-koordinater kallas också *kartesiska koordinater,* och är vanligt förekommande inom matematik för att ange positioner i grafer. X följer grafen horisontellt, medan Y följer grafen vertikalt.

<make_frame name=frame_name x_position=400 y_position=250>

När en ram finns, behövs ett fönster. Fönster har tagen *window*, och är en av de tags som kräver både start-tag och slut-tag. Det speciella med *window*, är att den fungerar som en egen, virtuell fil som går att ladda upp i en ram. All kod och text som befinner sig mellan start- och slut-tagen är endast aktiv i fönstret.

<window> <end window>

Ett fönster utan tagar eller text är tomt, och för att överhuvudtaget användas måste fönstret sättas i en ram – laddas upp i ramen. För att ladda upp ett fönster i en ram används attributet *load_into*. Värdet på *load_into* är den ram fönstret ska laddas upp i.

<window load_into=frame_name>

Om ett fönster inte har någon *load_into*, går det bara att komma åt genom en länk till fönstret. För att göra det med tagen *link* så behöver fönstret en variabel, som sätts med attributet *name*. Variabeln används sedan istället för en URL i länken.

<window name=my_window>

<link adress=my_window>

Följande attribut finns att tillgå i tagen *window:* load_into, name.

2.11 Gömd kod

Eftersom all kod i Caews kan skrivas i ett enda dokument, kan det ibland vara praktiskt att inaktivera bitar av kod, eller lägga in text som kommentarer som sedan inte kommer att synas i det färdiga resultatet. Detta kan vara särskilt praktiskt under testfasen, innan designen är färdig men det är onödigt att radera testkod.

För att gömma kod, används tagen *hide*. *Hide* fungerar på samma sätt som *text*, och behöver både en start-tag och en slut-tag.

<hide> <end tag>

Alla tags, all text etc. som befinner sig mellan *hide*'s start- och slut-tag hoppas över vid kompileringen av Caews kod.

3 Systemdokumentation

Systemdokumentationen över Caews presenterar hur språket översätts till språk en vanlig webbläsare förstår, grammatik, parser samt implementering.

3.1 Översättning

Nedan visas exempel på hur språket översätts till HTML, css samt Javascript. Kod i css är skriven med röd text, HTML är skriven med blå text och Javascript med orange text. All css-kod finns i en separat css-fil. All HTML och Javascript läggs i en fil med samma namn som Caews-filen koden skrevs i. All Caewskod är markerad med grått.

<background>

```
body {
background-color: #FFFFFF;
background-repeat: no-repeat;
}
```

<background name=my_background_settings image=/images/mybg.jgg fixed=yes transparancy=50>

```
my_background_settings {
background-color: #FFFFFF;
background-image: (‘/images/mybg.jpg’);
background-repeat: no-repeat;
background-opacity: 0.5;
}
```

```
<my_background_settings> All kod samt text som befinner sig på ytan där denna bakgrund är.
</my_background_settings>
```

<image image=/images/my_picture.png>

```
<img src=/images/my_picture.png>
```

<text font=Calibri size=12><end text>

```
text {
font-family: ’Calibri’;
font-size: 12px;
}
```

```
<text> Här skrivs texten i HTML-dokumentet.</text>
```

<text name=my_font font=Calibri size=12><end text>

```
my_font {
font-family: ’Calibri’;
font-size: 12px;
}
```

<my_font> Här skrivs texten i HTML-dokumentet.</my_font>

<link adress=http://my_page.com/ text=”min länk”>

```
a:link {
text-decoration: none;
color: #000000;
}
a:visited {
text-decoration: none;
color: #000000;
}
a:hover {
text-decoration: none;
color: #000000;
}
a:active {
text-decoration: none;
color: #000000;
}
```

Min länk

3.2 Grammatik

<tag> ::=	”<”, <tagtype>, <attributes>, ”>”
	”<”, <variable>, <attributes>, ”>”
	”<”, ”make”, <object type>, <attributes>, ”<”
	<boxtype>
<tagtype> ::=	[background, image, link, window, text, switch, make_frame]
<boxtag> ::=	<boxstart>, <tag>||<tags>, <content>, <boxend>
<boxstart> ::=	”<”, <tagtype>, <attributes>, ”>”
<boxend> ::=	”<”, ”end”, <tagtype>, ”>”
<attributes> ::=	<attribute>||<attributes>
<attribute> ::=	<keyword>, ”=”, <value> || <keyword>, ”=”, <variable>
<keyword> ::=	[A-Z,a-z]+
<value> ::=	<integer>||<constant>||<textstring>||<percent>
<integer> ::=	[0-9]+

<constant> ::= [A-Z,a-z]+ | <keyword>

<variable> ::= [A-Z, a-z]+

<textstring> ::= ”(”, [\.]+, ”)” || ”, [\.], ”

<content> ::= <tag>||<tags>||<boxtag>

3.3 Parsning

class Caews

```
#Only parses the first loop (i.e. body). Several loops are needed for each type of base tag.
def initialize
  @caewsParser = Parser.new("Caews") do
    token(/\s+/)
    token(/[<]/) {|m| m}
    token(/[>]/) {|m| m}
    token(/[=]/) {|m| m}
    token(/\d+/) {|m| m.to_i }
    token(/\w+/) {|m| m }
    token(/[A-Z+][a-z+][\.\?\!\,\:\;+]+/) {|m| m} #För all text inom <text>-tags.

  start :tags do
     match(:tag, :tags) {|tag, tags| "body {\n #{tag} \n #{tags} }"}
     match(:tag) {|tag| "#{tag}" }
  end


                rule :tag do
     match('<', :texttag, :attributes, '>', :texts, '<', 'end', 'text', '>') {|_, texttag, attrs, _, text, _, _, _, _|
Caews.new.print_to_file("#{texttag} {\n#{attrs}\n} \n Start of #{texttag}: \n\n #{text} \n\n end of
#{texttag}")}
                match('<', :tagtype, :attributes, '>') { |_, tagtype, attrs, _|
"#{tagtype}#{attrs}\n"}
     match('<', :tagtype, '>') {|_, tagtype, _| "#{tagtype} \n {}"}
                match('<', 'end', :tagtype, '>') { |_, _, tagtype, _| "</#{tagtype}>"}
                match('<', :name, :attributes, '>') { |_, tagname, attributes, _|
"#{tagname}#{attributes}"}

                end

   rule :texttag do
     match('text') {|a| "font"}
   end

   rule :tagroot do
                match('background') {|a| a}
     match('image') {|a| a}
```

```
  match('switch') {|a| a}
  match('link') {|a| "a"}
  match('make_frame') {|a| a}
  match('window') {|a| a}
 end

        rule :tagtype do
                match('background') {|a| a}
                match('image') {|a| a}
                match('switch') {|a| a}
                match('link') {|a| "a"}
                match('make_frame') {|a| a}
                match('window') {|a| a}
        end

        rule :attributes do
                match(:attribute) {|attr| "#{attr};"}
                match(:attribute, :attributes) {|attr, attrs| "#{attr}\n #{attrs}"}
#               match() {}
        end

        rule :attribute do
                match(:key, '=', :value) {|a, _, b| "-#{a}: #{b}"}
                match(:key, '=', :keyword) {|a, _, b| "-#{a}: #{b}"}
        end

        rule :key do
                match('name') {|a| a}
                match('colour') {|a| "color"}
                match('image') {|a| "src"}
                match('transparency') {|a| "opacity"}
                match('width') {|a| "width"}
                match('height') {|a| "height"}
                match('fixed') {|a| "fixed"}
                match('stretch') {|a| a}
                match('font') {|a| "family"}
                match('size') {|a| "size"}
                match('repeat') {|a| "repeat"}
                match('sleep') {|a| a}
                match('fade') {|a| a}
                match('adress') {|a| "href"}
                match('type') {|a| a}
                match('border') {|a| "border"}
                match('x_position') {|a| a}
                match('y_position') {|a| a}
                match('scroll') {|a| "scroll"}
                match('load_into') {|a| a}
        end

        rule :value do
                match(:integer) {|a| a}
                match(:keyword) {|a| a}
```

```
                match(:variable) {|a| a }
                match('"', :string, '"') {|_, a, _| a}
        end

        rule :variable do
  match(:string) {|_, _, a| a}
        end

    rule :text do
      match(/[A-Z+][a-z+][\.\?\!\,\:\;+]+/) {|a| a}
    end

    rule :texts do
      match(:text, :texts) {|a, b| "#{a}+#{b}"}
      match(:text) {|a| a}
    end

        rule :string do
                match(/w+/) {|a| a}
        end

        rule :keyword do
                match('yes') {|a| "repeat"}
                match('no') {|a| "no-repeat"}
                match('blue') {|a| "#3366FF"}
                match('red') {|a| "#CC0000"}
                match('black') {|a| "#000000"}
                match('pink') {|a| "#FFCC66"}
        end

        rule :integer do
                match(/\d+/) {|a| a}
        end
   end
end
#  def done(str)
#    ["quit","exit","bye",""].include?(str.chomp)
#  end

#       def run(file_name)
        def run(str)
#               str = File.open(file_name, 'r')
                $p = @caewsParser.parse str
                $html = File.new("main.html", "a")
                $css = File.new("main.css", "a")
      Caews.new.print_to_css($p)
        end

        def log(state = true)
                if state
                        @caewsParser.logger.level = Logger::DEBUG
                else
```

```
                                        @caewsParser.logger.level = Logger::WARN
                                end
                        end

                        def print_to_css(content)
        css_clear = File.open("main.css", "w")
        css_clear.write("Cleared file of old content.\n\n")
        css_clear.close()

        File.open("main.css", "a") do |css_file|
        css_file.write("#{content}")

         end
                        end

def print_to_html(content)
       html_clear = File.open("main.html", "w")
        html_clear.write("Cleared file of old content.\n\n")
        html_clear.close()

        html_file = File.open("main.html", "a")
        html_file.write("<link rel=\"stylesheet\" type=\"text\/css\" href=\"main.css\">")
 end

                        end

#Tester
CaewsTest = Caews.new()
#CaewsTest.run("test.cws")
#CaewsTest.run("<background colour=pink>")
CaewsTest.run("<background colour=pink><background colour=black>")
#CaewsTest.run("<text colour=black>hej hopp har ar lite text i ett textblock<end text>")
end #End Caews class
```

4 Reflektion

Att bygga ett språk var betydligt svårare – och lättare – än jag väntat mig. Det svåraste under den här tiden har varit själva tankearbetet och mina begränsade kunskaper om programmering. Hur ska jag lösa det här problemet? Eftersom arbetet har skett från grammatik-nivå, så har projektet behövt dissekeras ned till sina beståndsdelar för att sedan sättas ihop igen. En del saker har inte fungerat alls som väntat, vilket resulterat i en del konstiga fel.

I början undrade jag om jag möjligtvis tagit mig vatten över huvudet med min projektidé, eftersom den var väldigt annorlunda mot vad de övriga i klassen producerade. Dessutom krävde projektet att jag fördjupade mig i css, något som det var svårt att hinna med. Tid har varit en bristvara, något som är förargligt men inte kan hjälpas. Det har gjort det besvärligt att arbeta med projektet.

Ett av de problem jag stötte på under vägens gång, som också ställde till mycket problem för mig, var just hur Caews skulle sköta själva översättningen till html och css. Jag använder mig av en lexer och parser (något som möjligtvis kan ha varit ett felaktigt val som endast krånglade till saker), och det var väldigt svårt att konstruera generella lösningar för språket. Det största problemet var just det faktum att Caews inte bara skulle generera ett språk – utan två. Hade Caews klarat att bara översättas till exempelvis css eller html hade projektet blivit betydligt enklare, men eftersom bägge behövdes ställde det till oerhört mycket problem.

Jag tror att det här projektet kommer att bli mycket användbart för mig i framtiden. Inte nödvändigtvis för att jag kommer att använda mig av Caews personligen, utan på grund av att vi just plockat ned språken och därför skaffat en djupare förståelse för hur ett enkelt datorspråk fungerar.

I ett projekt som detta är det självklart att planeringen som satts upp tidigt förändras under resans gång, och så blev det också för mig. Jag fick göra många ändringar, Caews som slutprodukt och Caews som första utkast ser väldigt olika ut. Jag lyckades aldrig, inom tidsramen, implementera alla delar av Caews som jag först tänkt. Det fungerar i den mån att det går att kompilera och skapar css- och htmlfiler, men allt fungerar inte. Två saker som jag i dagsläget saknar lösning på är dels variabler, som blev mycket krångligt med parsern, samt att vanlig text inte kan skrivas i vilken form som helst. Bland annat blir det problem när det finns ett kommatecken någonstans i en mening. Helt oformaterad text och siffror utan kommatecken, punkter eller andra symboler fungerar.

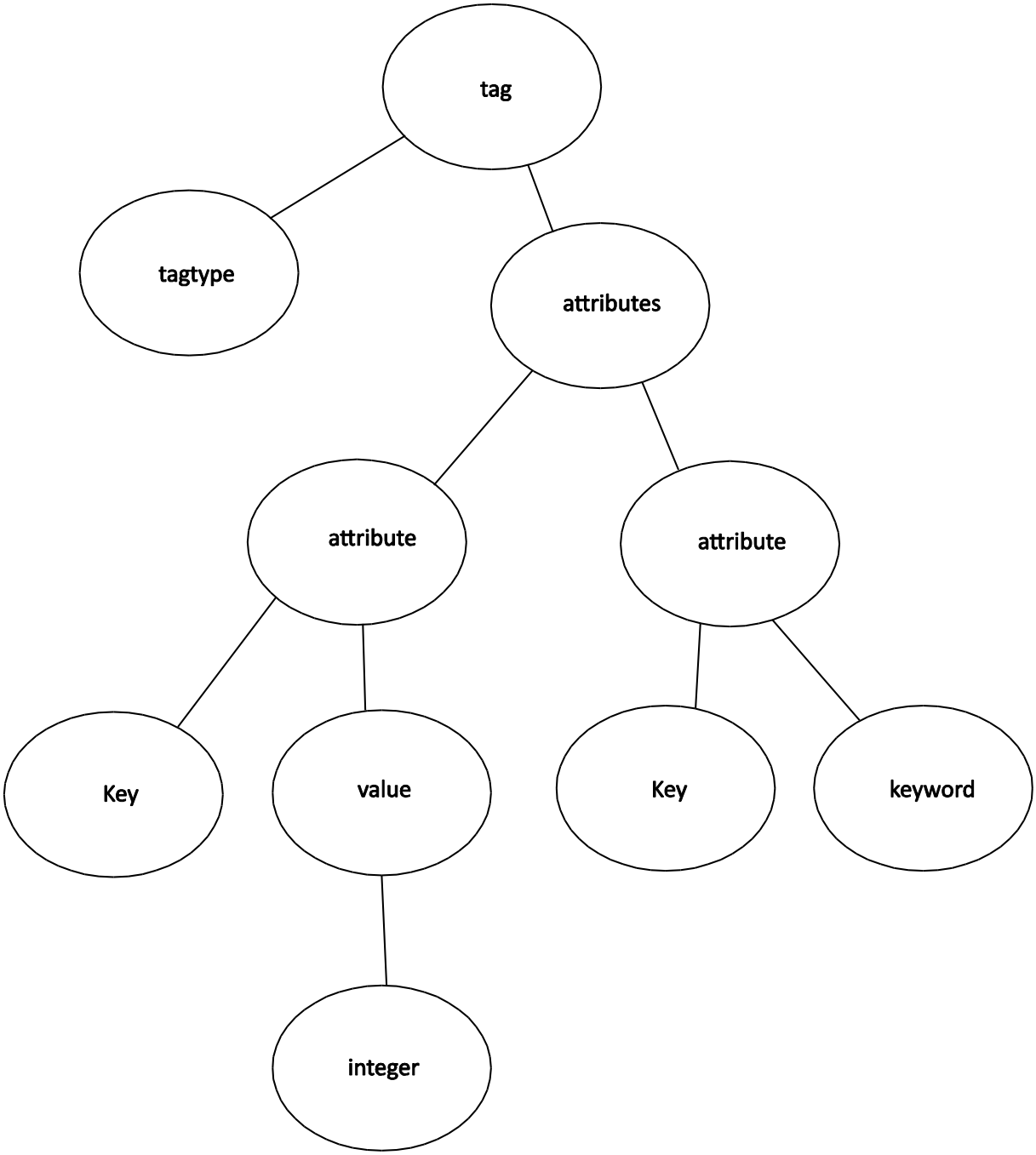
Enkel Caewskod fungerar alldeles utmärkt att skriva, Caews genererar gärna kod för exempelvis bakgrundsfärg. Jag insåg när det var dags för implementering att jag nog varit en smula överambitiös även om Caews i sig inte var särskilt komplicerat att implementera dök det upp andra problem som var besvärliga att lösa och tog lång tid.

Caewskoden som finns i denna rapport kan inte läsa in en fil med hjälp av caews_run.rb, som det från början var tänkt. Jag har helt enkelt fått prioritera bort det, även om caews_run.rb är en fullt fungerande fil för att ladda upp filer. Den har helt enkelt inte blivit sammankopplad med caews_parse.rb.

5 Bilagor

5.1 Syntaxträd

Exempel på ett syntaxträd för en enkel tag. Symboler utelämnade.



4.2 Hexadecimaltabell