COMS W4115 Language Reference Manual
CSSLang

Wayne Yang (wy2149@columbia.edu)

June 25, 2010

1 Introduction

This manual describes **CSSLang** language which implements a subset of the syntax of CSS 2.1 specification, and it follows closely with the descriptions of CSS syntax on http://www.w3.org/TR/CSS2/ and the SASS directives syntax described on http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html.

2 Lexical Conventions

A CSSLang source file has file extension **.csslang** and it contains one or more CSS *rules*. The CSSLang compiler transforms a **.csslang** file into a valid CSS file by carrying out variable expansions and directives introduced by lines beginning with the @ character. Any valid **CSSLang** source file can be compiled into a valid CSS 2.1 file, but not vice versa.

2.1 Tokens

There are four classes of tokens: identifiers, values, and operators.

2.2 White space

Characters "space" (U+0020), "tab" (U+0009), "line feed" (U+000A), "carriage return" (U+000D), and "form feed" (U+000C) are considered "white space" in **CSSLang**. White space and comments are ignored unless they separate tokens.

2.3 Comments

The characters /* introduce a comment and the characters */ terminate a comment. Comments do not nest.

2.4 Identifiers

CSSLang supports a subset of *identifiers* in CSS specification 2.1. In **CSSLang**, *identifiers* are sequences of characters [a-zA-Z], digits [0-9], hyphen (-), and the underscore (_). They can not start with a digit, or a hyphten followed by a digit. *identifiers* can appear as variable names, template names, property names, and selectors.

3 Selectors

In CSS, a selector is a pattern matching rule which determines which elements to apply the style rule to. The version of selector syntax implemented by **CSSLang** can be described in the following Yacc-like grammar:

```
type_selector:  identifer;
universal_selector:  '*';
id_selector:  '#' identifier;
class_selector:  '.' identifier;
simple_selector:  type_selector
   | universal_selector
   | id_selector
   | class_selector
   | type_selector id_selector
   | type_selector class_selector;
selector:  simple_selector
   | selector white_space simple_selector
```

4 Rule sets

A rule set is a selector followed by a declaration block. A declaration block begins with a left curly brace ({) and ends with a matching right curly brace (}). A declaration block can contain zero or more pairs of property name and value. The property name and value are separated by a colon (:) and terminated by a semicolon (;). A property name is an identifier. The syntax of values will be described in details in section *Values*.

The following is the format of a rule set:

```
rule_set:
   selector left_brace properties right_brace
properties:
   property_name_value
   | properties property_name_value
property_name_value:
   property_name colon property_value semicolon
```

The following is an example of a rule set:

```
div.header {
    float: left;
    color: #FFFFFF;
}
```

5 Values

CSSLang supports four main types of values: numbers, strings, colors, and booleans. The values can appear on the right-hand side of variable assignment.

5.1 Numbers

Numbers can have integer values or floating values. An integer constant consists of a sequence of digits and is taken to be decimal. A floating constant consists of an integer part, a decimal point, and a fraction part. Either the integer part or the fraction part (not both) may be missing. Numbers may be preceded by a "-" or "+" to indicate the sign. -0 is equivalent to 0. Numbers may be immediately followed by a unit. There are two types of units: relative and absolute. Relative units are **em**, **ex**, and **px**. Absolute units are **in**, **cm**, **mm**, **pt**, and **pc**. For details on these length units, please refer to the CSS 2.1 specification.

Only numbers of the same relative unit can appear as operands of an arithmetic operator. Conversion happens when numbers of different absolute units appear in an arithmetic expression. The unit of the left operand will be used as the base unit for conversion.

5.2 Boolean

There are two Boolean values: **true** and **false**.

5.3 Colors

A color value can be one of the predefined colors or a RGB speficiation. The list of predefined colors is: **aqua**, **black**, **blue**, **fushsia**, **gray**, **green**, **lime**, **maroon**, **navy**, **olive**, **organge**, **purple**, **red**, **silver**, **teal**, **white**, and **yellow**. An RGB value can be in either hexadecimal or functional notation and it is case-insensitive. The hexadecimal notation is a '#' immediately followed by either three or six hexidecimal characters. The functional notation is 'rgb(' followed by a comma-separated list of three integer values or three percentage values followed by ')'.

The examples below are colors in RGB hexadecimal notation:

```
#f11
#fff111
#FFFFFF
```

The examples below are colors in functional notation:

```
rgb(255,211,0)
rgb(100%, 33%, 0%)
```

5.4 Strings

String literals can be specified with double quotes ("), single quotes ('), or without quotes. If string literals are quoted, they must have matching quotes.

6 Operators

6.1 Parentheses ()

Parentheses are used to group expressions. A parenthesized expression is a primary expression whose type and value are identical to those of the original expression.

6.2 Negation operator (!)

Negation operator is an unary operator that converts a non-false operand to **true**, and **false** operand to **true**.

6.3 Arithmetic operators (+ - * /)

Arithmetic operators are binary operators that must have operands that can be evaluated to numeric values. Division by 0, and all floating-point exceptions are treated as error by the **CSSLang** compiler. Operands of different units are subject to the conversion mentioned in the *Numbers* sectcion.

6.4 Relational operators (< <= > >=)

Relational operators only work for operands that can be evaluated to numbers. Operands of different units are subject to the conversion mentioned in the *Numbers* sectcion.

6.5 Equality operators (== !=)

Equality operators work with all types of values. The operator == converts two operands to **true** if and only if both operands have the same unit and values. Conversion does not happen for expressions with equality operators.

6.6 Precedence & Associativity

The table below summarizes the rules for precedence and associativity of all operators. Operators on the same line have the same precedence; rows are in decreasing precedence:

OPERATORS	ASSOCIATIVITY
()	left to right
!	right to left
* /	left to right
+ −	left to right
< <= > >=	left to right
== !=	left to right

7 Expressions

Expressions are identifiers, strings, numbers or expressions in parentheses. Expressions are evaluated according to the associativity and precedence of operators involved. Parentheses can be used to change the order of evaluations in expressions.

Primary expressions are identifiers, values, or expressions in parentheses.

8 Variables

Variables are identifiers preceded with a dollar sign ($). Variables are declared outside of a rule set and the values are set using the same syntax as the properties:

```
$variable-name : expressions semicolon
```

The values of variables persist after the variable declarations until the end of the file. Variables inside rule sets and templates, and on the right-hand side of assignments are substituted with their values.

9 Directives

CSSLang adds support for directives that allow conditional styles and rule templates. Here are the directives: @if, @else, @def, and @inc.

9.1 @if and @elseif, and @else

The @if directive is used inside a rule set or a rule template and is used to choose style declarations depending on the result of expression:

```
@if expression {
   properties
} @elseif {
   properties
} @else {
   properties
}
```

If the expression is not evaluated to **false**, then the first declaration block is used. Otherwise the next immediate expression in the @elseif is evaluated until @else is reached.

9.2 @def and @inc

@def is used to define templates for rule sets. The templates can take parameters which are given the values when the template is included using the directive @inc. A template is declared with a template name follwed by parentheses and a declaration block. The parentheses can contain a comma-separated list of parameters in variable notation:

```
@def template_name($param1, $param2) {
   properties
}
```

A template can be used inside a declaration block of rule sets or templates by using the @inc directive followed by the template name and its parameters in parentheses.

```
selector {
   @inc template_name($param1, $param2);
}
```

@inc directive should be terminated by a semicolon.