

Confetti Specification

Version 0.7

Table of Contents

| | |
|---|----|
| Introduction | 2 |
| Conformance | 3 |
| Lexical Structure | 4 |
| Forbidden Characters | 4 |
| White Space | 4 |
| Comments | 4 |
| Line Terminators | 4 |
| Reserved Punctuators | 5 |
| Directive | 5 |
| Argument | 5 |
| Escaped Characters | 6 |
| Line Continuation | 6 |
| Bidirectional Control Characters | 6 |
| Grammar | 7 |
| Language Extensions | 8 |
| Parsers | 9 |
| Normative References | 10 |
| Annex A: Comment Syntax Extension | 11 |
| Annex B: Expression Arguments Extension | 12 |
| Annex C: Punctuator Arguments Extension | 13 |

This specification defines **version 0.7** of Confetti.

NOTE

Confetti is stable, however, user feedback *may* evolve the language. If you discover bugs in this specification or have ideas for alterations or new features, then please suggest them on the [projects issue tracker](#).

Introduction

This clause is informative.

Confetti is a configuration language intended for human-editable configuration files. It is minimalistic, untyped, and unopinionated.

This specification defines the core language for Confetti, but does not assign semantic meaning to it. Assigning semantic meaning is the responsibility of the Confetti user.

Confetti is designed to be minimal and extensible. Similar to how various flavors of Markdown exist, implementations can introduce custom Confetti extensions by extending the grammar. Official extensions can be found in the annex of this specification. These extensions provide a syntactic framework for extending Confetti in a way compatible with its grammar.

The Confetti grammar has been designed to allow each directive to be processed individually; that is, an implementation may process all arguments of a single directive before continuing to the next directive or subdirectives. This design allows for implementations that let the user decide if processing should continue based on their semantic interpretation of the directive.

The complete grammar is available in the back matter of this specification.

End of informative text.

Conformance

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC 2119](#). However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except examples and sections explicitly marked as informative.

Lexical Structure

Confetti source text consists of zero or more Unicode scalar values.

The Confetti language consists of zero or more *directives*. A directive consists of one or more *arguments* and optional subdirectives.

Forbidden Characters

Forbidden characters are Unicode scalar values with general category **Control**, **Surrogate**, and **Unassigned**. Forbidden characters must not appear in the source text.

White Space

White space characters are Unicode characters with the **Whitespace** property, excluding **line terminators**.

Comments

Comments must begin with a **#** and continue until a **line terminator** or EOF is found.

All Unicode characters should be allowed in comments excluding **forbidden characters**.

Example: A simple comment.

```
# This is a comment.
```

Line Terminators

Line terminators are defined by the Unicode standard and are listed in the following table.

For compatibility with Windows operating systems, implementations may treat the sequence Carriage Return (U+000D) followed by Line Feed (U+000A) as a single, indivisible new line character sequence.

| Name | Description |
|------|------------------------------|
| LF | Line Feed (U+000A) |
| VT | Vertical Tab (U+000B) |
| FF | Form Feed (U+000C) |
| CR | Carriage Return (U+000D) |
| NEL | Next Line (U+0085) |
| LS | Line Separator (U+2028) |
| PS | Paragraph Separator (U+2029) |

Reserved Punctuators

The following table lists all reserved punctuators.

| Name | Description |
|------|------------------------------|
| " | Quotation Mark (U+0022) |
| # | Number Sign (U+0023) |
| ; | Semicolon (U+003B) |
| { | Left Curly Bracket (U+007B) |
| } | Right Curly Bracket (U+007D) |

Directive

A directive is a command or statement.

A simple directive must be a sequence of one or more [arguments](#). A simple directive must be terminated by a character from the [line terminator](#) character set or a Semicolon (U+003B) character.

A compound directive must be a sequence of one or more [arguments](#), zero or more [line terminators](#), and subdirectives. Subdirectives shall be a sequence of zero or more directives enclosed by curly braces. Subdirectives begin with a Left Curly Bracket (U+007B) and terminate with a Right Curly Bracket (U+007D). A compound directive may be succeeded by a Semicolon (U+003B) character.

Argument

An argument shall be a sequence of one or more characters from the argument character set. The argument character set shall consist of any Unicode scalar value excluding characters from the [white space](#), [line terminator](#), [reserved punctuator](#), and [forbidden](#) character sets.

The value of an argument shall be the argument's lexeme with [escaped characters](#) processed.

Quoted Argument

A quoted argument is a directive argument enclosed by Quotation Mark (U+0022) characters.

Single quoted arguments must begin with a leading punctuator Quotation Mark (U+0022), followed by zero or more single quoted argument characters, and terminated by the trailing punctuator Quotation Mark (U+0022). The single quoted argument character set is the union of the argument character set and [white space](#) character set.

Triple quoted arguments must begin with three consecutive leading punctuator Quotation Mark (U+0022), followed by zero or more triple quoted argument characters, and terminated by the trailing punctuator Quotation Mark (U+0022). The triple quoted argument character set is the union of the argument character set, the [white space](#) character set, and the [line terminator](#) character set.

The value of a quoted argument shall be the argument's lexeme with [escaped characters](#) processed, leading and trailing Quotation Mark (U+0022) characters removed, and [line continuations](#) processed.

Escaped Characters

When a Reverse Solidus (U+005C) appears in an argument, a single quoted argument, or a tripled quoted argument, it must be succeeded by a Unicode character 'C'. Unicode character 'C' is any Unicode scalar value except [white space](#), [line terminators](#), and [forbidden characters](#).

The Reverse Solidus (U+005C) and character 'C' are together labeled as an *escaped character*. An escaped character is interpreted as if the character 'C' replaces the reverse solidus and 'C' character.

The escaped character shall not be interpreted as a [reserved punctuator](#) in any context.

Line Continuation

Within the Quotation Marks (U+0022) of a single quoted argument, when a Reverse Solidus (U+005C) immediately precedes a [line terminator](#), the implementation shall delete the reverse solidus and line terminator and continue processing single quoted argument characters.

Example: Line continuation in a quoted argument.

```
message "Hello, \ ①  
World!"
```

① Directive with arguments 'message' and 'Hello, World!'

When the last argument of a directive is a Reverse Solidus (U+005C) succeeded by a [line terminator](#), the implementation shall delete the reverse solidus and line terminator and continue processing arguments for the directive.

Example: Multi-line directive.

```
probe-device eth0 \ ①  
eth1
```

① Directive with arguments 'probe-device' and 'eth0' and 'eth1'

Bidirectional Control Characters

Implementations may reject Confetti source text with Unicode bidirectional formatting characters.

Grammar

The formal grammar:

```
directives = { <simple-directive> | <compound-directive> | <newline-char> }
simple-directive = <arguments> ( <newline-char> | ';' )
compound-directive = <arguments> { <newline-char> } <subdirectives> [ ';' ]
subdirectives = '{' <directives> '}'
arguments = <argument> { <argument> }
argument = <simple-argument> | <quoted-argument>
simple-argument = <argument-char> { <argument-char> }
quoted-argument = <single-quoted> | <triple-quoted>
single-quoted = '"' { <argument-char> | <space-char> } '"'
triple-quoted = '"""' { <argument-char> | <space-char> | <newline-char> } '"""'
```

The lexical grammar:

```
argument-char = [ ^ \p{Whitespace} \p{gc=Control} \p{gc=Surrogate}
\p{gc=Unassigned} \u0022 \u0023 \u003B \u007B \u007D ]
newline-char = [ \u000A \u000B \u000C \u000D \u0085 \u2028 \u2029 ]
space-char = [ [ \p{Whitespace} ] - [ \u000A \u000B \u000C \u000D \u0085
\u2028 \u2029 ] ]
```

Language Extensions

Implementations may extend this specification, provided that any additions are clearly documented.

Parsers

An implementation may set limits on the size of the Confetti source text that it accepts. An implementation may set limits on the maximum nesting depth of subdirectives. An implementation may set limits on the number of Unicode scalar values an [argument](#) can have. An implementation may set limits on the number of arguments a directive can have. An implementation may set limits on the number of subdirectives a directive can have.

Normative References

This specification is written against the Unicode Standard version 16.0. It is possible newer standards may necessitate a revision of this specification.

This specification defines formal grammar rules using EBNF as defined in ISO/IEC 14977:1996.

This specification defines lexical grammar rules using Unicode character set notation as defined in Unicode Technical Standard #35.

Annex A: Comment Syntax Extension

Implementations may include a comment syntax based on C language conventions where multi-line comments are enclosed in `/*` and `*/` and single-line comments begin with `//`.

Single-line comments must behave identically to Confetti `#` comments.

Valid multi-line comments must begin with a `/*` and continue until a `*/` is found.

The following table extends the [reserved punctuator](#) table.

| Name | Description |
|-----------------|----------------------------------|
| <code>//</code> | Two Solidus (U+002F U+002F) |
| <code>/*</code> | Solidus Asterisk (U+002F U+002A) |
| <code>*/</code> | Asterisk Solidus (U+002A U+002F) |

Annex B: Expression Arguments Extension

Implementations may support *expression arguments*. An expression argument is a user-defined expression that may appear wherever an [argument](#) is allowed. The interpretation and evaluation of an expression argument is implementation defined.

This extension requires Left Parenthesis (U+0028) and Right Parenthesis (U+0029) to be interpreted as [reserved punctuators](#). If an implementation permits parenthesis in the expression argument, then they must be *balanced*. Parentheses are balanced if for every Left Parenthesis (U+0028) there is a succeeding Right Parenthesis (U+0029).

The production rules shall be amended as follows:

```
argument ::= <simple-argument> | <quoted-argument> | <expression-argument>
expression-argument ::= '(' ? implementation defined ? ')'
```

Annex C: Punctuator Arguments Extension

Implementations may define their own *punctuator arguments*. A punctuator argument is a self-delimiting argument that may appear wherever an [argument](#) is allowed.

Punctuator arguments shall consist of one or more characters from the [argument character set](#).

Punctuator arguments must be self-delimiting; that is, they are treated as a single argument. Adjacent characters must not contribute to the punctuator argument, even if they would under standard interpretation.

The production rules shall be amended as follows:

```
argument ::= <simple-argument> | <quoted-argument> | <punctuator-argument>
punctuator-argument ::= ? implementation defined ?
```