1.0 | Overview

This document describes version 0.0.2-alpha of Gridline's Grid binary file format.

The Grid File is the native file format for spreadsheets within Gridline. The purpose of the Grid file is to describe the graphical appearance of spreadsheets and store cell contents in a cross-platform and cross-application manner.

2.0 | Syntax

This chapter discusses the data representation and structure of a Grid file on an object and file scope.

2.1 | Character Set

Grid files, on an atomic level, comprise a sequence of 8-bit bytes. Its character set is a subset of the American Standard Code for Information Interchange (ASCII). There are three classes of characters: *text*, *delimiters*, and *whitespace*. These classes apply to all characters in the character set except those within higher-level entities (see Section 2.2); in such cases, separate rules apply.

Whitespace characters (see Table 2.1) separate contiguous *lexemes* that would otherwise combine to form a single *token*. Some whitespace characters, may additionally serve as markers denoting the end of a line. Such characters are called *newline* characters.

Table 2.1: Whitespace characters with their respective representations.

Dec	Hex	Symbol	Name
09	09	HT	Horizontal Tab
10	0A	LF	Line Feed
13	0D	CR	Carriage Return
32	20	SP	Space

When a Carriage Return character (CR) immediately precedes a Line Feed character (LF), it will be recognized as a single newline character, which indicates an End of Line (EOL) marker. By themselves, the CR and LF characters are also newline characters. However, they do not, by themselves, constitute an EOL marker.

Delimiter characters (see Table 2.2) denote the start and end of a higher-level entity such as a string or formula. Like whitespace characters, delimiter characters separate sequences of characters that will otherwise combine to form a *token*. Delimiters should be balanced such that every open delimiter has a corresponding closing delimiter of the same type. These characters are not part of the entities they terminate or instantiate.

Table 2.2: Delimiter characters with their respective representations.

Dec	Hex	Symbol	Name
36	24	$	Dollar Sign
37	25	%	Percent Sign
40	28	(	Open Parenthesis
41	29	)	Close Parenthesis
60	3C	<	Less Than
62	3E	>	Greater Than

Text characters are all printable characters except whitespace and delimiters.

2.2 | Tokens

A *lexeme* is the smallest meaningful element in a syntax. They are the made up of the maximum sequence of contiguous characters that satisfy the conditions established by the initial character. Conditions determine the bounds of the lexeme and the type of *token*. A *token* is an comprised of a lexeme and its corresponding type. A lexeme type and token type are synonymous.

A token type can be categorized into two classes: *data*, and *non-data.* Tokens belonging to the *data* class contain textual or numerical values, which can be represented by the following token types:

- *Identifier*
- *Literal*

Unlike their counterparts, *non-data* types do not hold values. Rather, they tether data token types together or allow them to be distinguishable from eachother. Such tokens can be represented by the following types:

- Delimiter

2.2.1 | Identifiers

An *Identifier* is a distinctive lablel used to refer to token types of the data class. It is a sequence of one or more alphanumeric characters of arbitrary length. A dollar sign character ($) must be the first character and it must be followed by a letter. While the dollar sign is not part of the identifier name, itself, it does indicate that the following sequence of text characters are. Uppercase and lowercase characters are distinct such that $x and $X are not the same. The following are examples of valid identifiers:

```
$n
$HelloWorld
$C2x2
```

2.2.2 | Integer Literals

An *integer literal* is a sequence of digits representing a numerical constant as a whole number. An integer literal can be represented in decimal and hexadecimal notations. These representations may also have leading zeros, but they must follow their prefix (if any).

An integer literal represented by a decimal number is the default representation. Decimal numbers do not have a corresponding prefix denoting its base. However, they can be prefixed with an negative sign (-) to denote a negative number. Decimal numbers are comprised of a sequence of ASCII digits (0-9). Decimal numbers may also use underscores (_) to break the sequence into more readable chunks. An underscore must not immediately follow another underscore.The following are examples of valid decimal integer literals:

```
172
-55
00010
-0025
1_000_000
-24_250
```

An integer literal represented by a hexadecimal number is comprised of a `0x`- prefix followed by a sequence of ASCII digits (0-9) and letters (A-F and a-f). Hexadecimal numbers may be also use underscores (_) to break the sequence into more readable chunks. An underscore must not immediately follow another underscore. Hexadecimal representations should not be prefixed with a negative sign.

The following are examples of valid integer literals:

```
0x12a4
0xfabC
0x1aaa_0000
0x0001_fade
```