@ ? d: -> =>

Lonelog: Dice Notation Add-on

A Standard Grammar for Recording Dice Expressions

Version 1.1.0



Table of contents

Overview **1**
What This Add-on Adds 2
Design Principles 2
Part I: Practical Reference **3**
How dice expressions fit into a log entry 3
Rolling Dice 5
Adding a Flat Modifier 6
Keeping or Dropping Dice 6
Exploding Dice 7
Dice Pools (Counting Successes) 9
Roll Descriptions 10
Common Patterns by System Type 10
Quick Reference **13**
Dice Types 13
Modifiers 13
Compare Point Operators 14
Output Flags (Tool Implementations) 14
Common Patterns 15
FAQ **16**
Appendix: Complete Grammar Reference **18**
A.1 Dice Types 18
A.2 Modifiers 19
A.3 Group Rolls 29
A.4 Roll Descriptions 31
A.5 Math 33
Credits & License **35**

Overview

Core Lonelog gives you `d:` to mark a mechanics roll. What goes inside that field is left to you. That freedom is intentional — different systems use wildly different dice conventions — but it creates friction when logs are shared or when tools try to parse them.

This add-on solves that by defining a standard grammar for dice expressions. It is adapted from the specification used by rpg-dice-roller, an open-source dice library, and extended with Lonelog-specific conventions for how expressions appear inside `d:` entries.

The notation is a superset of the informal standard described in the Wikipedia article on Dice Notation. If you already write `2d6+3` or `4d6dl1`, you are already using it. This document formalizes that habit and fills in the gaps.

Who needs this add-on:

- Players who share session logs and want their dice expressions to be unambiguous
- Developers building tools that parse or render Lonelog logs
- Players using systems with complex dice mechanics (dice pools, exploding dice, Fudge/Fate) who want a consistent way to write them

Who can skip it: If you play a single system and never share logs, your informal dice notation probably works fine. Core Lonelog does not require this add-on.

What This Add-on Adds

This add-on does not introduce new Lonelog tags or structural blocks. It standardizes the dice expression grammar that appears inside existing core elements, primarily `d:` roll fields.

Element	Purpose	Example in a log
Standard expression	A dice roll with optional modifiers and math	`d: Attack 2d6+3=9 -> Hit`
Roll description	Inline label attached to a dice expression	`d: 4d6dl1 // Ability score=14`
Group roll	Multiple sub-expressions evaluated together	`d: {4d6, 2d8}k2=11`

No new core symbols. This add-on introduces no additions to `@`, `?`, `d:`, `->`, or `=>`.

Design Principles

Grammar, not engine. This spec defines how to *write* dice expressions, not how to *resolve* them. Any compliant tool should produce equivalent results, but the notation itself is system-agnostic.

Upward compatible. Simple expressions — `d6`, `2d10+3` — are valid without reading this document. Complexity is opt-in.

Log-readable. Expressions are meant to be read by humans in plain text. Modifier flags (`!`, `k`, `d`) are chosen to be short and memorable, not just machine-parseable.

Part I: Practical Reference

This section covers the dice expressions that come up most often in solo RPG play. If you are logging sessions rather than building a parser, this is all you need.

How dice expressions fit into a log entry

Core Lonelog (§3.2.1) gives the `d:` line its order: `d: [Label] <dice>[+mod]=<total> [vs <target>] -> <outcome>`. This add-on fills in the `<dice>` part. The dice expression records *what was rolled* and `=N` records the *numeric total*. The `->` field is reserved for the interpreted outcome — a state, not a number.

```
d: Strike 2d6+2=9 -> Hit
d: Pick the lock d20+3=17 vs DC 15 ->
Success
d: Overcome 4dF+2=3 -> Success with Style
```

Individual results go in brackets directly after the expression, before `=`:

```
d: Strike 2d6+2[4, 3]=9 -> Hit
d: Ability 4d6dl1[6, 4d, 5, 3]=15 -> STR
15
```

The breakdown is optional. Include it when the individual results matter — a lucky max roll, a dropped die that changed the outcome — or leave it out when the total is all you need.

Roll context (core §4.1.9), the tags that help or hinder a roll, goes *before* the expression as `+[…]` and `-[…]`. Brackets before the dice are context; brackets after them are results:

```
d: Strike +[flanking] -[wounded] 2d6+2[4,
3]=9 -> Hit
```

Rolling Dice

The core syntax is `NdS`, where `N` is the number of dice and `S` is the number of sides. `N` defaults to `1` if omitted.

```
d6       roll one 6-sided die
2d6      roll two 6-sided dice and sum
the results
4d10     roll four 10-sided dice and sum
the results
d20      roll one 20-sided die
d%       roll percentile dice (1–100);
equivalent to d100
dF       roll one Fudge/Fate die (-1, 0,
or +1)
4dF      roll four Fudge/Fate dice and
sum the results
```

In a log:

```
d: Strike 2d6=9
d: Oracle d%=34
d: Fate 4dF=1
```

Adding a Flat Modifier

Use standard arithmetic. The modifier applies to the total, not to individual dice.

```
d6+3     roll d6, add 3
2d6-1    roll 2d6, subtract 1
d20+5    roll d20, add 5
```

In a log:

```
d: Strike 2d6+2=9
```

Keeping or Dropping Dice

These are the most common modifiers in solo play. Use them for advantage/disadvantage, ability score generation, or any system that rolls and discards dice.

Keep highest / lowest:

```
2d20kh1    roll 2d20, keep the highest
(advantage)
2d20kl1    roll 2d20, keep the lowest
(disadvantage)
4d6kh3     roll 4d6, keep the highest 3
```

`k` alone defaults to highest: `2d20k1` is the same as `2d20kh1`.

Drop lowest / highest:

```
4d6dl1     roll 4d6, drop the lowest
(ability scores)
4d6dh1     roll 4d6, drop the highest
```

`d` alone defaults to lowest: `4d6d1` is the same as `4d6dl1`.

In a log:

```
d: Ability score 4d6dl1=15
```

The breakdown (`4d6dl1[6, 4d, 5, 3]=15`) is optional — useful when you want to record which die was dropped, but not required.

Note: `k` (keep) and `d` (drop) are the same operation expressed from opposite directions. `4d6kh3` and `4d6dl1` both give you the top 3 dice from a pool of 4.

Exploding Dice

An exploding die is re-rolled when it hits its maximum value, and the new roll is added to the total.

```
d6!      roll d6; re-roll and add if you
roll a 6
2d6!     roll 2d6; each die that rolls 6
explodes
```

In a log:

```
d: Damage 2d6! =13
```

> ***Note:*** *Whenever an expression contains a compare point or ends with a modifier that accepts one (*`!`*,* `r`*,* `u`*,* `cs`*,* `cf`*, or a pool target like* `>=7`*), write a space before the* `=` *that records the result.* `2d6!=13` *would be parsed as "explode on any roll not equal to 13";* `2d6! =13` *records a result of 13. The same goes for pools:* `10d6=6 =2` *means "count the sixes; there were two".*

You can change the trigger with a compare point:

```
2d6!>4    explode on any roll greater
than 4
4d10!<=3  explode on any roll less than
or equal to 3
```

For systems where exploded values stack into a single result (e.g. Savage Worlds raises):

```
d6!!     compound exploding: re-rolls
stack on the same die
```

Dice Pools (Counting Successes)

Some systems count how many dice beat a target rather than summing results. Attach a compare point directly to the die notation:

```
5d6>=5     count dice showing 5 or higher
4d10>7     count dice showing 8, 9, or 10
10d6=6     count dice showing exactly 6
```

In a log, the result is the success count. Because the expression contains a compare point, put a space before the `=` that records it:

```
d: Attack 5d10>=7 =3 -> Hit
d: Sixes 10d6=6 =2 -> Two hits
```

To count failures as negative successes, add an `f` compare point immediately after:

```
4d6>4f<3   successes on 5–6, failures on
1–2
```

Roll Descriptions

Attach a label to a dice expression for clarity, especially when logging multiple rolls on one line.

Use `//` for inline descriptions:

```
4d6dl1 // Strength
2d6!   // Damage

d: 4d6dl1 // STR=15, 4d6dl1 // DEX=12,
4d6dl1 // CON=9
```

For a single roll, the label at the start of the `d:` line does the same job and is easier to read: `d: Strength 4d6dl1=15`. Descriptions earn their place when several expressions share one line.

Common Patterns by System Type

D&D / OSR ability scores:

```
d: Stats 4d6dl1 // STR=15
```

Usage Dice (The Black Hack, Macchiato Monsters):

```
d: Supply d8=2 -> Step down!
d: Torchlight d4=1 -> Depleted!
```

Note: A usage die is written as a standard die expression. The step-down mechanic (`d8→d6`) and resource tracking are covered in the Resource Tracking Add-on.

PbtA / Ironsworn 2d6:

```
d: Clash 2d6+2=9 -> Weak Hit
```

Advantage / Disadvantage:

```
d: Persuade 2d20kh1+4=16 vs DC 15 ->
Success
d: Sneak    2d20kl1+6=15 vs DC 16 -> Fail
```

Fate / Fudge:

```
d: Overcome 4dF+2=3 -> Success with Style
```

Savage Worlds trait roll — the trait die and the Wild Die both ace, and you keep the higher. Note the space before `=`:

```
d: Fighting {d8!, d6!}kh1 =12 -> Hit
```

Percentile oracle:

```
? Is the gate guarded? (50/50)
d: d%=47 vs 50 -> Yes
```

Quick Reference

Dice Types

Notation	Meaning
`d{n}`	Standard die with n sides
`d%`	Percentile die (1–100)
`dF` / `dF.2`	Standard Fudge die
`dF.1`	Variant Fudge die

Modifiers

Notation	Name	Order
`min{n}`	Minimum	1
`max{n}`	Maximum	2
`!` / `!{cp}`	Exploding	3
`!!` / `!!{cp}`	Compounding	3
`!p` / `!!p`	Penetrating	3
`r` / `ro` / `r{cp}`	Re-roll	4
`u` / `uo` / `u{cp}`	Unique	5
`kh{n}` / `kl{n}`	Keep highest / lowest	6
`dh{n}` / `dl{n}`	Drop highest / lowest	7
`{cp}`	Target success	8
`f{cp}`	Target failure	8
`cs` / `cs{cp}`	Critical success	9

Notation	Name	Order
`cf` / `cf{cp}`	Critical failure	10
`s` / `sa` / `sd`	Sort	11

Compare Point Operators

Operator	Meaning
`=`	equal to
`!=`	not equal to
`<>`	not equal to (use with exploding)
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

Output Flags (Tool Implementations)

Flag	Meaning
`!`	Exploded die
`d`	Dropped die
`*`	Success
`_`	Failure
`**`	Critical success
`__`	Critical failure
`^`	Min modifier applied
`v`	Max modifier applied

Common Patterns

```
2d6+3        PbtA / Ironsworn stat roll
2d20kh1      Advantage (D&D 5e)
2d20kl1      Disadvantage (D&D 5e)
4d6dl1       Ability score (D&D)
4dF+2        Fate roll with skill
d%           Percentile oracle
2d6!         Exploding roll
5d10>=7      Dice pool, target 7+
4d6>4f<3     Success/failure pool
```

FAQ

Q: Does this replace the notation I already use in my logs?
A: No. If your existing dice notation is unambiguous to you, keep using it. This spec is a standard to adopt when consistency matters — for sharing logs or building tools.

Q: What's the difference between `4d6dl1` and `4d6kh3`?
A: They produce identical results — roll 4d6, keep the top 3. `dl1` (drop lowest 1) and `kh3` (keep highest 3) are two ways to express the same operation. Use whichever reads more naturally for your system.

Q: Can I use these expressions anywhere in a Lonelog entry, or only in `d:` fields?
A: The grammar applies wherever you record a dice result. It's most meaningful in `d:` fields, but nothing prevents using it in `=>` narrative lines or session notes when noting what was rolled.

Q: My system uses a custom die (like a d3 or d7). Is that valid?
A: Yes. Any positive integer is a valid number of sides: `d3`, `d7`, `d13`, `d100`. The standard dice shapes are just the most common cases.

Q: How do I record a roll result in my log — just the expression, or the full breakdown?
A: The expression and total are always sufficient: `d: Attack 2d6+2=9 -> Hit`. The bracketed individual results (`d: Attack 2d6+2[4, 3]=9 -> Hit`) are optional — include them when the breakdown is meaningful (a die that exploded, a dropped result that mattered), skip them when it isn't. The reference examples in the appendix show breakdowns in the style of a dice roller's output (`2d6!: [4, 6!, 6!, 2] = 18`); in a log line, move the brackets after the expression as shown here.

Q: The `d` modifier letter conflicts with the `d` in `NdS`. How does a parser tell them apart?

A: By position. In `4d6`, the `d` separates quantity from sides. In `4d6d1`, the second `d` follows a complete die expression and is therefore the Drop modifier. Context disambiguates them.

Q: What about dice that aren't rolled — oracle tables, card draws, etc.?
A: This notation covers dice only. Oracle answers, table results and non-dice mechanics are recorded with core Lonelog's `?`, `d:`, `tbl:` and `->` fields in free-form text. The dice grammar is a layer on top of core notation, not a replacement for it.

Appendix: Complete Grammar Reference

This appendix is the full dice-roller grammar the add-on is built on. Most logs never need it: Part I covers what solo play actually uses. Come here when your system does something unusual, or when you are building a tool.

A.1 Dice Types

Note: A single die type has a minimum quantity of `1` and a maximum quantity of `999`. Valid: `d8`, `1d10`, `999d6`. Invalid: `0d10`, `1000d6`, `-1d20`.

Standard

Notation: `d{n}`

A standard die has a positive integer number of sides.

```
d6    roll a single 6-sided die
4d10  roll a 10-sided die 4 times and sum
the results
```

Percentile

Notation: `d%`

Rolls a whole number between 1 and 100. Shorthand for `d100`.

```
4d%    equivalent to 4d100
```

Fudge / Fate

Notation: `dF` / `dF.2` / `dF.1`

Six-sided dice marked with `-`, `+`, and blank faces, corresponding to `-1`, `+1`, and `0`.

- `dF` or `dF.2` — standard: two of each face
- `dF.1` — variant: four blanks, one `+`, one `-`

```
dF    standard Fudge die; equivalent to
dF.2
dF.1  variant Fudge die
4dF   roll four standard Fudge dice and
sum
```

A.2 Modifiers

Modifiers alter the dice pool after rolling. Multiple modifiers can be combined; they execute in a fixed order (listed below) regardless of the order written.

```
5d10!k2    explode first (order 3), then
keep highest 2 (order 6)
5d10k2!    equivalent — order is always
by modifier priority
```

Min

Notation: `min{n}` | **Order:** 1

Any roll below the minimum is treated as the minimum value.

```
4d6min3: [3^, 4, 3, 3^] = 13
```

Note: Statistically, rolling the minimum becomes more likely than any other single value. `d6min3` gives a 50% chance of rolling 3.

Max

Notation: `max{n}` | **Order:** 2

Any roll above the maximum is treated as the maximum value.

```
4d6max3: [3v, 3v, 3, 2] = 11
```

Exploding

Notation: `!` / `!{cp}` | **Order:** 3

When a die rolls its maximum value, it is re-rolled and the new roll is added to the total. The re-roll can also explode.

```
2d6!: [4, 6!, 6!, 2] = 18
```

The second die rolled 6 and exploded; the re-roll also rolled 6 and exploded again.

Use a compare point to change the trigger:

```
2d6!=5    explode on 5
2d6!>4    explode on 5 or 6
4d10!<=3  explode on 1, 2, or 3
```

> ***Note:*** *The* `!=` *(not equal) operator conflicts with the exploding notation. Use* `<>` *instead:*
>
> ```
> 2d6!!=4   wrong — this means
> "compound on 4", not "explode on
> not-4"
> 2d6!<>4   correct — explode on any
> value that is not 4
> ```

Note: To prevent infinite loops, modifiers are limited to 1000 iterations per die. `1d10!>0` returns at most 1001 rolls.

Compounding

Notation: `!!` / `!!{cp}`

Like exploding, but re-rolls are added to the original die's total rather than listed separately.

```
2d6!:  [4, 6!, 6!, 2] = 18   exploding:
separate rolls
2d6!!: [4, 14!!]       = 18
compounding: combined into one
```

Penetrating

Notation: `!p` / `!!p` / `!p{cp}` / `!!p{cp}`

Like exploding, but each re-roll has 1 subtracted from it. From the HackMaster Basic rules.

```
2d6!p: [6!p, 5!p, 5!p, 3, 1] = 20
```

Actual rolls were `[6, 6, 6, 4, 1]`; each exploded re-roll has 1 subtracted.

Re-roll

Notation: `r` / `ro` / `r{cp}` / `ro{cp}` | **Order:** 4

Re-rolls a die until it rolls above the minimum value, discarding the previous roll (unlike exploding, which keeps all rolls).

```
d6r      re-roll 1s indefinitely
d6ro     re-roll once on a 1, even if the
re-roll is also 1
2d6r=5   re-roll any die showing 5,
indefinitely
4d10r<=3 re-roll any die showing 1, 2, or
3
```

Unique

Notation: `u` / `uo` / `u{cp}` / `uo{cp}` | **Order:** 5

Re-rolls any die whose result duplicates another in the pool.

```
2d10u    re-roll duplicates until all
results are unique
2d10uo   re-roll duplicates once; may
still have duplicates
4d6u=5   only re-roll duplicates that
show 5
```

Note: If the number of dice exceeds the number of sides (e.g. `5d3u`), unique results are impossible for all dice. The modifier is limited to 1000 iterations to prevent infinite loops.

Keep

Notation: `k{n}` / `kh{n}` / `kl{n}` | **Order:** 6

Keeps only the highest or lowest N results; all others are discarded. The opposite of Drop.

`k` alone defaults to highest.

```
4d10kh2  keep the highest 2
4d10k2   equivalent to above
4d10kl1  keep the lowest 1
```

Discarded dice are marked with `d` in tool output:

```
6d8k3: [3d, 6, 7, 2d, 5, 4d] = 18
```

Drop

Notation: `d{n}` / `dh{n}` / `dl{n}` | **Order:** 7

Drops the highest or lowest N results. The opposite of Keep.

`d` alone defaults to lowest.

```
4d10dl2    drop the lowest 2
4d10d2     equivalent to above
4d10dh1    drop the highest
4d10dh1dl2 drop both the highest and
lowest 2
```

Note: *Both modifiers look at the full dice pool, including dice already flagged by the other modifier. Using them together can override each other unexpectedly.*

```
3d10k1dh1: [7d, 1d, 2d] = 0   drops
everything
3d10k1d1:  [6d, 1d, 9]  = 9   keeps
only the highest
```

Target Success / Dice Pool

Notation: `{cp}` | **Order:** 8

Counts how many dice meet a condition rather than summing values. Append a compare point directly to the die notation.

```
5d10>=8              count dice showing 8
or higher
2d6=6: [4, 6*] = 1  one success
4d3>1: [1, 3*, 2*, 1] = 2
```

Successful dice are marked with *.

Note: *A target compare point immediately following a modifier that uses compare points is interpreted as the modifier's compare point, not a success target. Specify the target first:*

```
2d6!>3    explode on any roll
greater than 3
2d6>3!    explode on 6; greater than
3 is a success
```

Target Failure / Dice Pool

Notation: `f{cp}` | **Order:** 8

Must follow a Target Success modifier directly. Each failure subtracts 1 from the success count.

```
4d6>4f<3: [2_, 5*, 4, 5*] = 1
```

Failed dice are marked with `_`.

Critical Success

Notation: `cs` / `cs{cp}` | **Order:** 9

Marks a die as a critical success. Aesthetic only — does not change the numeric result.

```
2d20cs: [4, 20**] = 24
4d10cs>7            rolls greater than 7
are critical successes
```

Critical success dice are marked with `**`.

Critical Failure

Notation: `cf` / `cf{cp}` | **Order:** 10

Marks a die as a critical failure. Aesthetic only.

```
2d20cf: [4, 1__] = 5
4d10cf<3           rolls less than 3 are
critical failures
```

Critical failure dice are marked with `__`.

Sorting

Notation: `s` / `sa` / `sd` | **Order:** 11

Sorts the dice results before display. Ascending by default.

```
4d6:   [4, 3, 5, 1]
4d6s:  [1, 3, 4, 5]   ascending (default)
4d6sa: [1, 3, 4, 5]   ascending
(explicit)
4d6sd: [5, 4, 3, 1]   descending
```

Compare Points

A compare point is an operator followed by a number. Used by Exploding, Re-roll, Unique, Target Success, Target Failure, Critical Success, and Critical Failure modifiers.

Operators:

Operator	Meaning
`=`	equal to
`!=`	not equal to
`<>`	also not equal to (use with exploding)
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

```
d6!=3    explode on any roll not equal to
3
d10!>=5  explode on 5, 6, 7, 8, 9, or 10
d4r<3    re-roll anything less than 3
```

Note: *Both mean "not equal to" and are interchangeable in most contexts. However,* `!=` *conflicts with the exploding notation —* `2d6!!=4` *is parsed as "compound on 4", not "explode on not-4". Use* `<>` *when combining not-equal with exploding:*

```
2d6!<>4   correct: explode on any
value that is not 4
```

A.3 Group Rolls

Group rolls evaluate multiple sub-expressions and apply modifiers to the group as a whole, rather than to individual dice. Sub-expressions are comma-separated and enclosed in `{}`.

```
{4d6, 2d10, d4}: {[2, 6, 4, 2], [7, 4],
[1]} = 26
```

Without a modifier, sub-roll totals are summed. A single sub-roll in `{}` is functionally equivalent to the expression without braces.

Keep and Drop with Groups

When applied to a group with multiple sub-rolls, `k` and `d` keep or drop entire sub-rolls by their total value.

```
{4d10, 5d6, 2d10}k2   keep the 2 sub-
rolls with the highest totals
{4d10, 5d6, 2d10}d2   drop the 2 sub-
rolls with the lowest totals
```

When applied to a single sub-roll, they keep or drop individual die results within the expression:

```
```

```
{4d10*(2+5d6)}k2   keep the highest 2
individual rolls across all 9 dice
```

Target Success / Failure with Groups

Compares each sub-roll's *total* against the compare point rather than individual dice.

```
{3d20+5}>40: {([11, 7, 19]+5)*} = 1

{4d6+2d8, 3d20+3, 5d10+1}>40:
{[4,3,3,2]+[2,6], ([17,5,20]+3)*,
([7,9,6,10,8]+1)*} = 2
```

Sorting with Groups

Sorts both individual dice within each sub-roll and sub-rolls by their total.

```
{4d6/2, 3d4+3, 2d10}s    ascending
{4d6/2, 3d4+3, 2d10}sd   descending
```

A.4 Roll Descriptions

A textual label can be attached to any dice expression or group roll. Labels are not parsed as notation.

Inline Descriptions

Use `//` after the expression:

```
4d6dl1 // Strength score
2d6!   // Damage roll
{4d6, 2d10, d4} // Fire, frost, falling
```

In a Lonelog `d:` field:

```
d: 4d6dl1 // STR=14, 4d6dl1 // DEX=11
```

The rpg-dice-roller grammar also accepts `#` for inline descriptions. **Lonelog logs don't use it**, because `#` marks reference tags in core (`[#N:Jonah]`).

Block Descriptions

Use `/* ... */` for multi-word or multi-line labels:

```
4d6 /* Ice damage */
{5d6+5} /* You can even
write on multiple lines */
```

The rpg-dice-roller grammar also accepts `[ ... ]` for block descriptions. **Lonelog logs don't use it**, because brackets after a dice

expression hold its individual results, and brackets before it hold roll context.

Any dice notation inside a description block is not parsed:

```
4d6 /* Ice damage 5d10+7 */   only 4d6 is
rolled
```

A.5 Math

Arithmetic Operators

Standard operators apply to roll totals:

Operator	Meaning	Example
`+`	add	`2d6+3`
`-`	subtract	`d20-2`
`*`	multiply	`d6*5`
`/`	divide	`2d10/d20`
`%`	modulus (remainder)	`d15%2`
`^` or `**`	exponent	`3d20^4`

Operators can also determine how many dice to roll or how many sides they have:

```
(4-2)d10   roll a d10 twice (4-2=2)
3d(2*6)    roll a 12-sided die (2*6=12)
three times
```

Parentheses

Parentheses group expressions and define order of operations:

```
1d6+2*3:   [4]+2*3 = 10    multiplication
first
(1d6+2)*3: ([4]+2)*3 = 18  addition first
```

Credits & License

© 2026 Roberto Bisceglie

This add-on extends Lonelog by Roberto Bisceglie.

The dice notation grammar and the reference in the appendix are adapted from the rpg-dice-roller specification by GreenImp Media Limited, used under the MIT License. The original MIT license and copyright notice are reproduced below.

MIT License
Copyright (c) 2020–2021 GreenImp Limited
Copyright (c) 2021–present GreenImp Media Limited

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Version History:

- v1.1.0: The complete grammar moves to an appendix; the space before a result's = applies to every expression with a compare point; Savage Worlds example corrected (trait die and Wild Die, keep the higher). Aligned with Lonelog v1.7. Individual results go in brackets directly after the expression; roll context goes before it. # and [...] descriptions are not used in Lonelog logs. Example fixes.
- v1.0.0: Initial release

This work is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License**.

You are free to share and adapt this material, provided you give appropriate credit and distribute adaptations under the same license. Session logs and play records created using this add-on's notation are your own work and are not subject to this license.