



Aid4Mail Search List Syntax

User Guide

Fookes Software Ltd
Charmey, Switzerland
www.aid4mail.com

Aid4Mail Search List Syntax

Basic Rules

- Create a plain text file with .txt extension
- Place one search term per line
- Do not use double quotes or parentheses unless searching for those literal characters
- Do not use Boolean operators (AND, OR, NOT), search operators, or comments within the list
- Group search terms by prefixing lines with '+' or '-' (refer to the user guide for details)
- Leave blank lines between related terms for readability (Aid4Mail ignores them)
- Lines starting with a semicolon (;) or a hash (#) are treated as comments and are ignored during processing
- Regular expressions are based on the PCRE2 syntax

Wildcards and Modifiers (Listed by Performance)

Fast Performance

- Plain-text terms (fastest when 20+ characters)
- * – Matches zero or more characters within a word (e.g., `corrupt*` matches corrupt, corruption, corrupted)
- ? – Matches exactly one character (e.g., `organi?e` matches organise and organize)
- # – Matches zero or one non-alphanumeric character (e.g., `data#breach` matches data breach, data-breach, databreach)
- `{[R]=pattern}` – Simple regular expressions, like `{[R]=\b(inappropriate|unwelcome|unwanted)\b}`

Medium Performance

- ~ – Performs stemming when placed at end of word if dictionary is set (e.g., `steal~` matches steal, steals, stole, stolen)

Slower Performance (Listed from faster to slower)

- `<+n>` – Matches up to n words between terms, in specified order (e.g., `money<+3>laundering`)
- `<n>` – Matches up to n words between terms, in bidirectional order
- `<+. >` – Matches terms in same sentence, in specified order
- `<. >` – Matches terms in same sentence, in bidirectional order
- `<+* >` – Matches terms in same paragraph, in specified order
- `<* >` – Matches terms in same paragraph, in bidirectional order
- `{[R]=pattern}` – Complex regular expressions, like `{[R]=\b[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,}\b}`

Primary Search Performance Rule

The list is processed from top to bottom. As soon as a match is found, remaining terms are skipped. Therefore:

1. Place terms most likely to be found at the TOP of the list
2. Place terms least likely to be found at the BOTTOM

Secondary Ordering (When Match Likelihood is Similar)

Order by search term complexity, from simplest to most complex:

1. Plain-text terms (fastest when 20+ characters)
2. Simple character wildcards (*, ?, #) and regular expressions
3. Stemming (~)
4. Word proximity wildcards (<n>, <. >, <* > and their ordered variants)
5. Very complex regular expressions

Best Practices

Avoid Common Terms

Avoid overly broad search terms that may produce irrelevant matches or false positives (e.g., date, meeting, news).

Avoid Redundancy

Don't include unnecessary variations of a search term. For example, if `offshore` doesn't match, a later occurrence of `offshore<+3>account` won't match either. In such cases, remove the more complex term to improve performance.

Group Plain Search Terms Into a Regular Expression

To improve efficiency, group plain search terms into a single regular expression instead of listing them individually. For example, instead of:

```
inappropriate
unwelcome
unwanted
```

Use:

```
{[R]=\b(inappropriate|unwelcome|unwanted)\b}
```

Optimal Use of Wildcards

- Use # instead of an apostrophe (e.g., `don#t` finds dont, don't, don't).
- Use stemming (~) and the multi-character wildcard (*) thoughtfully – they serve different purposes:

- Add ~ at the end of complete words to find inflected forms (e.g., `steal~` finds steal, steals, stole, stolen).
- Use * for prefixes/suffixes and partial matches (e.g., `corrupt*` finds corrupt, corruption, corrupted).

Limit Proximity Wildcards

Use word proximity wildcards sparingly:

- The specific-order wildcards, `<+n>`, `<+.>`, and `<+*>`, often give better results because the unordered versions, `<n>`, `<.>`, and `<*>`, can generate many false positives.
- Limit to 2–3 word proximity wildcards per search term. More than this can significantly impact performance, slow processing, and reduce efficiency.
- Consider breaking complex proximity searches into separate plain-text terms (e.g., `please<+3>enable<+3>editing` becomes `please enable editing` and `please enable document editing`).

Example (Ordered for Optimal Performance)

```
# Common business terms likely to appear frequently
{[R]=\b(meeting|report|update)\b}

# Industry-specific terms that appear regularly
data retention policy
{[R]=\b(proprietary|classified)\b}

# More specific terms using simple wildcards and stemming
corrupt*
fraud*
discriminate~

# Complex patterns that require more processing
money<+3>laundering
insider<.>trading
block*<3>promote~
{[R]=\b[A-Z0-9._%+~]+@[A-Z0-9.-]+\.[A-Z]{2,}\b}
```