Using recipes

The Underwriter recipe base: the library included with aggregate, the reproductions corpus and your own DecL

Stephen J. Mildenhall

2026-10-30

Every named DecL declaration is stored in a **recipe base**, a map (kind, name) ↦ `Recipe`. A recipe contains the parsed specification, the program, its `note`, `tags` and `hints`, and its source: the file it came from, or `'session'` for a program built in the current session. A later entry under the same key replaces an earlier one. An `Underwriter` holds a recipe base, and the module-level `build` is an underwriter that reads `library.agg`, the recipe library included with aggregate, on first use. A program consisting of a bare name builds the stored entry, and the references `agg.NAME`, `sev.NAME` and `dist.NAME` copy a stored entry into a new program.

The recipe base is held in memory. Nothing is written to disk until a call to `to_agg`, which writes to `~/.aggregate` by default, the folder that `Underwriter(databases='user')` reads. The code on this page writes only to a temporary folder and removes it at the end.

```
agg My.Book 50 claims sev lognorm 10 cv 1 poisson   # stored as (agg, My.Book)
My.Book                                             # builds the stored entry
port My.Port agg.My.Book agg.BasicBook              # agg.NAME copies in an entry
```

The package includes `reproductions.agg`[1], the DecL programs of the *Reproductions* book. They rebuild published actuarial and numerical results, and each is tagged with the paper it comes from. There is an analogous recitations corpus running through textbook exercises.

```
import tempfile
from pathlib import Path
import pandas as pd
from aggregate import build, qd, Underwriter, __version__
from style import wprint, GT, refuse, figures
figures()
print(__version__)
```

```
1.43.1
```

The library recipe

The recipe library included with aggregate, `library.agg`, is read on the first access to the recipe base of `build`. The code below shows how many entries it holds of each kind. The method `discover` filters a recipe base by a regular expression on the name, by `kind` and by `tags`; an entry matches only when it has every tag given.

[1] From aggregate 1.1.0

```
lib = build.recipes
print(len(lib), [p.name for p in build.databases])
wprint(lib.index.get_level_values('kind').value_counts().to_dict())
GT(build.discover(kind='agg', tags='role:intro').head(3))
```

```
206 ['library.agg']
{'agg': 120, 'port': 19, 'sev': 17, 'bvagg': 14, 'xpnl': 14, 'pnl': 9,
'distortion': 7, 'stable': 4, 'chfagg': 1, 'nttagg': 1}
```

name	program
BasicBiasedCoins	agg BasicBiasedCoins dfreq [1000] dsev [0 1] [0.51 0.49]
BasicBook	agg BasicBook 250 claims 1000 xs 0 sev lognorm 100 cv 1.5 poisson
BasicDfreq	agg BasicDfreq dfreq [1 2 3] dsev [1 2 10]

The method `recipe` returns one entry without building it, and its `as_read` attribute is the entry as it appears in its file, with its note and tags. The program `BasicBook`, a bare name, builds the entry.

```
wprint(build.recipe('BasicBook').as_read)
print()
qd(build('BasicBook'))
```

```
agg BasicBook
  250 claims
  1000 xs 0
  sev lognorm 100 cv 1.5
  poisson
  note{A basic realistic insurance book with Poisson frequency, lognormal
  severity and an occurrence limit. This is the workhorse compound
  distribution.}
  tags{topic:aggregate, role:hero, role:intro}

Aggregate object: BasicBook. Frequency distribution poisson. Severity
lognorm, [0, 1,000], bounded; 1,000 xs 0. Updated with bucket size 1 and
log2 = 16. Validation: not unreasonable.

       Mean     SD       CV     Skew   P01 Median   P99
X
Freq    250 15.811 0.063246 0.063246
Sev  98.137 128.91   1.3136   3.5092     4     55   693
Agg   24534 2561.7  0.10441  0.19862 18954  24449 30862
```

The reproductions corpus

The reproductions corpus is not in the default recipe base of `build`, so we read it into an underwriter of our own. An underwriter reads only the files given in its `databases` argument: a file name, a glob, a path, or one of `'default'`, `'user'` and `'all'`. It can be loaded by setting `databases='reproductions'`. Entries are named `<CitationKey>.<name>` and tagged `topic:<CitationKey>`, so a tag selects the programs from one paper.

```
rep = Underwriter(name='Reader', databases='reproductions')
papers = pd.Series([t for ts in rep.recipes.tags for t in ts]).value_counts()
print(len(rep.recipes), len(papers))
wprint(papers.head(5).to_dict())
GT(rep.discover(tags='topic:Gerber1982'))
```

```
143 30
{'topic:Bear1990': 27, 'topic:Mack2003': 17, 'topic:Jin2016': 14,
'topic:Pittarello2024': 13, 'topic:Liu2026': 7}
```

name	program
Gerber1982.Gerber	agg Gerber1982.Gerber 1 claim sev 2 * uniform + 1 poisson
Gerber1982.Gerber.v2	agg Gerber1982.Gerber.v2 10 claims sev 2 * uniform + 1 poisson
Gerber1982.Gerber.v3	agg Gerber1982.Gerber.v3 100 claims sev 2 * uniform + 1 poisson

The `recipes` frame of an underwriter has one row per entry. Its `note` column flags whether a note exists but omits the text, so a search of the notes goes through `recipe`. The program is a column, so `str.contains` searches it directly.

```
notes = pd.Series({n: rep.recipe(n, k).note for k, n in rep.recipes.index})
for name, note in notes[notes.str.contains('Exhibit|stop loss')].items():
    wprint(f'{name:21s} {note}')
progs = rep.recipes.program
wprint(progs[progs.str.contains('splice')].index.get_level_values('name').tolist())
```

```
Bodoff2017.B          @Bodoff2017, Exhibit 4's Pareto severity under a policy
                      limit.
Bodoff2017.B.v2       @Bodoff2017, Exhibit 4's Pareto severity under a policy
                      limit.
Gerber1982.Gerber     @Gerber1982, a compound Poisson on a uniform severity, for
                      the stop loss table.
Gerber1982.Gerber.v2  @Gerber1982, a compound Poisson on a uniform severity, for
                      the stop loss table.
Gerber1982.Gerber.v3  @Gerber1982, a compound Poisson on a uniform severity, for
                      the stop loss table.
Venter1983.Venter     @Venter1983, Exhibit 3, a compound Poisson over a
                      tabulated claim size law.
['Mack2003.Riebesell', 'Mack2003.Riebesell.v2', 'Denuit2025a.Denuit.Pool',
'Denuit2025a.Denuit.Pool.v2', 'Denuit2025a.Denuit.PoolAlt',
'Denuit2025a.Denuit.PoolB']
```

A corpus entry builds by name, like a library entry. `Gerber1982.Gerber` is a compound Poisson with one expected claim and a severity uniform on $[1, 3]$.

```
qd(rep('Gerber1982.Gerber'))
```

```
Aggregate object: Gerber1982.Gerber. Frequency distribution poisson.
Severity uniform, [0, 3], bounded. Updated with bucket size 1/1,024 and
log2 = 18. Validation: not unreasonable.

      Mean      SD      CV   Skew    P01 Median    P99
X
Freq     1       1       1      1
Sev      2 0.57735 0.28868      0 1.0195      2 2.9805
Agg      2  2.0817  1.0408 1.1086      0 1.7188  8.293
```

A scratch recipe base from fork

The method `fork` copies a recipe base in a few microseconds, sharing its parsed entries. What you build in a fork stays in the fork, and the recipe base of `build` never includes it. `load` and `fork` are

thread safe, but concurrent builds on one underwriter are not, so in a program with several threads give each thread its own fork.

```
mine = build.fork('Mine')
mine.build('agg My.Book 50 claims sev lognorm 10 cv 1 poisson '
           'note{a test book} tags{topic:mine}')
mine.build('port My.Port agg.My.Book agg.BasicBook')
print(('agg', 'My.Book') in mine.recipes.index,
      ('agg', 'My.Book') in build.recipes.index)
```

```
True False
```

The entries you build are those whose source is `'session'`. Their `seq` numbers follow those of the library entries, which are numbered from 0. The method `format_agg(pattern)` returns the session entries whose names match `pattern`, as DecL text.

```
display(GT(mine.recipes.query("source == 'session'")[['seq', 'tags', 'source']]))
wprint(mine.format_agg('My.Book'))
```

kind	*name*	seq	tags	source
agg	My.Book	206	('topic:mine',)	session
port	My.Port	207	()	session

```
agg My.Book 50 claims sev lognorm 10 cv 1 poisson note{a test book}
tags{topic:mine}
```

The method `preview` resolves the references in a program without building anything. Each reference is listed with its kind, its name and its source, here one session entry and one library entry.

```
for ref in mine.preview('port Both agg.My.Book agg.BasicBook').resolved:
    print(ref.kind, ref.name, getattr(ref.source, 'name', ref.source))
```

```
agg My.Book session
agg BasicBook library.agg
```

Saving recipes to a file and reading them back

The method `to_agg` writes the entries built in the current session to a `.agg` file. A relative name is written into `~/.aggregate` and an absolute path is used as given, so the absolute path below keeps this page out of your own recipe files. The `spread` layout puts each clause on its own line.

```
tmp = tempfile.TemporaryDirectory()
path = mine.to_agg(Path(tmp.name) / 'mine.agg', layout='spread')
wprint(path.read_text())
```

```
# written by aggregate 1.43.1 on 2026-10-30 12:33:43
# 2 program(s); pattern='.*', kind='all', source='session'
agg My.Book
  50 claims
  sev lognorm 10 cv 1
  poisson
  note{a test book}
  tags{topic:mine}
```

```
port My.Port
  agg My.Book
    50 claims
    sev lognorm 10 cv 1
    poisson
    note{a test book}
    tags{topic:mine}
  agg BasicBook
    250 claims
    1000 xs 0
    sev lognorm 100 cv 1.5
    poisson
    note{A basic realistic insurance book with Poisson frequency, lognormal
    severity and an occurrence limit. This is the workhorse compound
    distribution.}
    tags{topic:aggregate, role:hero, role:intro}
```

A new underwriter given the file as its `databases` reads it back. The reference `agg.BasicBook` in the portfolio was written out in full, so the file is self-contained.

```
back = Underwriter(name='Back', databases=str(path))
print(back.recipes.index.tolist())
qd(back('My.Book'))
```

```
[('agg', 'My.Book'), ('port', 'My.Port')]
Aggregate object: My.Book. Frequency distribution poisson. Severity
lognorm, [0, inf), subexponential right tail. Updated with bucket size
1/32 and log2 = 16. Validation: not unreasonable.

      Mean     SD      CV    Skew    P01 Median    P99
X
Freq    50 7.0711 0.14142 0.14142
Sev     10     10       1  3.9999 1.0312 7.0625 49.062
Agg    500    100     0.2 0.39999 295.31 493.62    762
```

Refused lookups and calls

A lookup of a name that is not in the recipe base raises `RecipeNotFound`, a subclass of `KeyError`. With the corpus loaded beside the library, some names appear under two kinds: the library defines the severity curves as `sev` entries and the corpus wraps each in an `agg`, so a lookup by name alone is ambiguous and needs `kind=`. `to_agg` does not overwrite an existing file unless `mode='w'`, and `Underwriter` accepts keyword arguments only. The last lines remove the temporary folder.

```
both = build.fork('Both')
both.load('reproductions')
refuse(lambda: build.recipe('No.Such.Book'),
       lambda: both.recipe('Mack2003.Lognorm'),
       lambda: mine.to_agg(path), lambda: Underwriter('reproductions'),
       kind=True)
print(both.recipe('Mack2003.Lognorm', kind='sev'))
tmp.cleanup()
print(path.exists())
```

```
RecipeNotFound: no recipe named 'No.Such.Book'
KeyError: 'Mack2003.Lognorm' is ambiguous across kinds (agg, sev); write
agg.Mack2003.Lognorm or sev.Mack2003.Lognorm, or pass kind= to choose. Shipped
library names are unique across kinds.
```

```
FileExistsError: <tmp>\tmpmibbvyvc\mine.agg already exists; pass mode="w" to
overwrite or mode="a" to append.
TypeError: Underwriter.__init__() takes 1 positional argument but 2 were given
Recipe('sev', 'Mack2003.Lognorm')
False
```