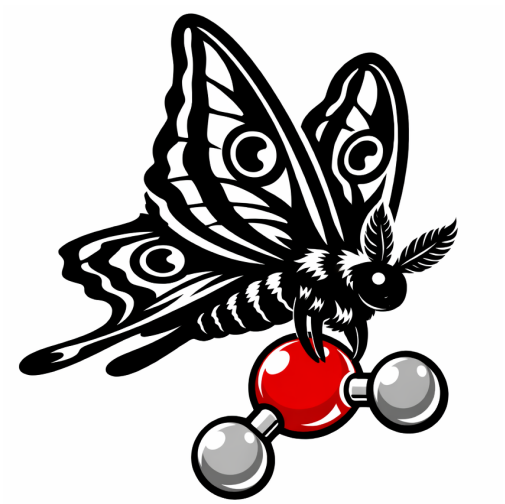

BaneTask v2.39.0

面向量子化学批量计算的任务描述、执行组织与结果管理系统

website: https://bane-dysta.top/software/banetask

Bane QC Project

目录

BaneTask 把量化化学计算、通用脚本和结果分析组织为可追踪的任务工作流。本手册说明任务文件、程序后端、运行配置、状态、结果、Derive、Plot、Process DSL 与命令行工具的当前行为。

手册正文按用户完成工作的顺序组织。正式语法只在一个位置完整定义；其他章节给出摘要和交叉引用。组合型工作流放在 `recipes/`；正文只定义用户接口、当前行为和可操作的诊断方法。

1 程序概览

1.1 BaneTask 处理的对象

BaneTask 读取 case 或 project 中的任务声明，解析依赖与变量，为每个任务生成程序输入和 workflow，交给本地进程、调度器或远程 target 执行，再把状态、结果和 artifact 写回项目。

对象	含义	常见载体
case	一组共享起始结构和任务文件的计算目录	一个含 `.bt` 与几何文件的目录
project	包含一个或多个 case 的项目入口	`.projbt` 或项目目录
task	一个可准备、运行和记录结果的计算步骤	`$task_name` 块
workflow	任务准备后生成的可执行描述	`.bwrk`、`pre_comd`、`comd`
result	程序输出、规范属性和用户 KV 的统一结果	快照、record、revision、数据库
artifact	与任务或项目关联的文件对象	manifest、归档对象、`.btdb`

1.2 BaneTask 与 BaneLib

BaneTask 调用 BaneLib 完成分子结构、量化程序输入、输出解析、规范属性和部分分析计算。用户在 BaneTask 中选择程序后端与任务字段，不直接调用 BaneLib 的内部接口。

1.3 命令行工具如何配合

工具	主要职责
`banetask`	解析任务文件、展开任务、准备输入和 workflow；也执行 Derive 与 Plot
`btrun`	选择 target，运行或提交已准备的 workflow，查询与取消队列任务
`btproc`	在 workflow 中执行结果抽取、Process DSL、FCclasses、数据库发布等处理步骤
`btdb`	查询、校验、报告、打包、恢复和导入结果数据库及 `.btdb` 归档
`btool`	创建模板、格式化、迁移、展开、解释和静态检查任务文件
`btc`	以交互或模板方式创建任务输入
`var`	读取、修改和导出 YAML、JSON 与任务文件变量

1.4 从任务文件到结果

处理顺序如下：

1. 读取配置、环境变量和任务文件。
2. 解析 YAML 头部、include、变量、matrix 与 foreach。
3. 建立任务、依赖和条件，确定 source。
4. 应用几何、电荷、自旋和其他修改。
5. 由程序后端生成输入、manifest、前处理、主命令和后处理。
6. 写出 `.bwrk` 和任务目录。
7. 由 `btrun` 在本地、调度器或远程 target 执行。
8. 更新 `bt.status`，抽取规范结果并发布 artifact。
9. 按声明运行 Derive、Plot 和项目归档。

1.5 最小 case

建立目录：

```
demo/
├── origin.xyz
└── job.bt
```

`origin.xyz`：

```
3
0 1
O  0.000000  0.000000  0.117790
H  0.000000  0.755453 -0.471161
H  0.000000 -0.755453 -0.471161
```

`job.bt`：

```
project: demo

$opt
  %source origin.xyz
  %program gaussian
  %control totcore 4
  %control totmem 8000
  %keywords B3LYP/6-31G(d) Opt
```

在 case 目录准备任务：

```
banetask job.bt
```

生成完成后运行可调度任务：

```
btrun kick --path .
```

只在当前机器运行一个已准备 workflow 时，可使用 `btrun task`；完整参数见第 6 章。

1.6 查看状态和结果

`bt.status` 记录任务准备与运行阶段。常用检查方式：

```
cat bt.status
btrun list --path .
btdb case summary --path .
```

任务目录中的主输出、`run.log`、`task.manifest.json`、`.bane.result.kv` 和结果处理日志用于进一步诊断。任务是否完成由 workflow 状态、阶段退出码和必要结果共同决定。

1.7 阅读路径

- 编写任务文件：第 2 章。
- 选择程序并确认生成物：第 3 章。
- 配置本地、调度器或远程运行：第 4 章。
- 查询结果、编写 Derive 或 Plot、创建归档：第 5 章。
- 查找指令、CLI、配置、状态和属性：第 6 章。

2 任务控制文件语法

本章定义任务文件的结构、展开阶段、任务块和全部任务指令。示例仅展示写法；参数、默认值、重复规则和错误行为以对应条目为准。

2.1 任务文件基础

本章说明 `.bt` 与 `.projbt` 的文件结构、YAML 头部、变量、矩阵、`&INCLUDE` 结构化并入和 `@foreach` 文本预展开。任务块中的 `%...` 指令在本章后半部分说明。

2.1.1 输入文件总览

BaneTask 的任务与项目入口包括：

- `.bt`：单 case 任务入口
- `.projbt`：项目入口

两者使用同一套 DSL 语法。一个任务文件由两部分组成：其一是用于描述项目级变量和元数据的 YAML 头部，其二是用于定义实际计算步骤的任务块。YAML 头部为全局声明，任务块是逐步执行的流程描述。

任务 DSL 的行内注释从引号外的 `#` 开始，但 `#` 只有位于行首或其前一个字符为空白时才是注释起始符。单引号或双引号中的 `#` 保留为普通字符；引号内使用反斜杠转义的下一个字符也按字面保留。因此 `${#array[@]}`、URL fragment 和不以空白分隔的 `name#tag` 不会被截断。需要开始注释时，写成 `...  # comment`。

下例展示文件头变量、matrix 和一个计算任务。matrix 会为 `basis` 的每个取值展开任务变体。

```
project: demo
define:
```

```
  method: B3LYP
matrix:
  basis: 6-31G(d), def2-SVP

$sp
  %source origin.xyz
  %program gaussian
  %keywords "[method]/[basis] SP"
```

已有 case 还可显式运行 `.btder` 和 `.btplt` 独立分析脚本；两种文件与同目录唯一的 `.bt` 配对。

2.1.2 YAML 头部

2.1.2.1 普通元数据

普通 `key: value` 项会作为头部元数据保存，并向后传递到：

- 结果快照导出启用时的 `.bane.taskmeta.json`
- 结果记录 JSON
- 数据库中的 `headerVariables`

常见元数据示例：

```
project: demo
owner: qm-group
SMILES: CCO
```

2.1.2.2 `autorun`

`autorun` 控制 workflow 结束后是否重新执行 `btrun kick`，以及哪些阶段错误可以被忽略。可用值如下：

值	续扫条件
`never`、`false`	不生成或调用独立续扫脚本。未写 `autorun` 时采用此行为。
`strict`	preprocess、主命令和 process 全部成功后才续扫。
`default`、`true`	preprocess 和主命令成功后续扫；process 失败会记录警告，但不阻塞后续任务。
`always`	无论 preprocess、主命令或 process 是否失败，都尝试续扫。

常用写法：

```
autorun: default
```

布尔写法 `autorun: true` 等价于 `autorun: default`，`autorun: false` 等价于 `autorun: never`。值不区分大小写；其他值会被视为无效配置并禁用 autorun。

process 阶段包括用户 `%process` 生成的后处理命令，以及追加到同一后处理脚本中的隐式结果导出。因此，在 `default` 模式下，结果导出器不支持某个文件格式时，任务会记录 `success_`

with_warnings, 但仍可继续触发下游任务。**always** 模式在普通阶段失败时记录 **continued_with_errors**。这两个状态都允许 **btrun** 继续解析依赖，同时保留失败阶段和退出码供检查。

always 只忽略任务阶段错误。任务被取消、收到终止信号、状态文件无法可靠更新或 workflow 未能生成时，不执行续扫。

启用 autorun 时，任务目录中会生成独立脚本：

- POSIX：**autorun_btrun**
- Windows：**autorun_btrun.bat**

该脚本优先使用 **BANETASK_BTRUN_PATH** 解析出的 **btrun**, 进入任务目录的上一级 case 目录，并执行：

```
btrun kick --backend auto --path .
```

调用过程默认记录到 **.banetask_autorun_btrun.trace**，可用 **BANETASK_AUTORUN_LOG** 改写日志路径。续扫脚本自身的返回码会写入日志，但不会替换 workflow 的主返回状态。

若当前 workflow 本身由 **btrun** 提交到 Slurm、SGE 或 PBS，生成的调度脚本会把本次 allocation 标记和每节点资源传给后续进程。因此 **autorun_btrun** 再次执行 **btrun kick** 时，可以把当前计算节点识别为 allocation-local target。普通后续任务仍按正常调度策略选择 target；只有其 **.bwrk** 带 **localfirst: true** 且当前 allocation 资源可立即取得时，才直接在该节点执行。

2.1.2.3 btrun

```
btrun:
  backend: slurm
  profile: beta
```

该块会被扁平化为 **btrun.backend** 与 **btrun.profile**，并写入生成的 **.bwrk** 头部，作为 **btrun** 选择 target/backend 与 profile 的路由提示。顶层 **backend**、**profile** 及其别名也可表达同一组路由信息。

2.1.2.4 资源默认值 totcore / totmem

文件头部可以用 **totcore** 和 **totmem** 为本文件中的任务指定默认资源：

```
totcore: 32
totmem: 96000
```

- **totcore** 表示默认总核心数。
- **totmem** 表示默认总内存，单位为 MB。
- 该默认值会应用到文件内尚未在 **%control** 中显式指定同名资源的任务。
- 单个任务块中的 **%control totcore** / **%control totmem** 优先级更高，可覆盖文件头部默认值。
- 这两个字段影响输入生成和 workflow 资源头部；local runner 的总资源上限仍由 **btrun** 配置或 local profile 决定。

例如，下列写法会让 **opt** 和 **td** 默认使用 32 核、96000 MB 内存，而 **sp** 单独改用 16 核：

```
totcore: 32
totmem: 96000

$opt
  %source origin
  %keywords "opt [method]/[basis]"

$td
  %source opt
  %keywords "td [method]/[basis]"

$sp
  %source opt
  %control
    totcore 16
  %keywords "sp [method]/[basis]"
```

2.1.2.5 finalize

finalize 用于在当前 case 或整个 project 的有效任务分支全部完成后，自动生成一个隐式的收尾脚本任务，用于统一执行数据库重建、Obsidian 导出、方法学说明生成和 FCclasses 速率汇总等只应在末次运行执行一次的动作。该能力由 YAML 头部元数据控制；未显式配置时默认启用。显式写法如下：

```
finalize: true
```

省略 **finalize** 或写成上式都会启用默认动作集：

- **method.generate**
- **db.rebuild**
- **obsidian.export**

当前任务文件包含有效的 FCclasses 任务时，系统还会在数据库重建前加入 **fcclasses.report**，把各任务的速率报告汇总到 **Results/FCclasses/**。

也可以使用结构化配置写法：

```
finalize:
  enabled: true
  scope: project
  actions:
    - method.generate:
        out: Results/Method/method.md
        json: Results/Method/method.json
        strict: true
    - db.rebuild
    - obsidian.export
  obsidian:
    out: ./vault/out
```

快捷布尔开关写法：

```yaml
finalize:
  enabled: true
  scope: case
  method:
    generate: true
    out: Results/Method/method.md
    json: Results/Method/method.json
    refs: method/references.db
  fcclasses:
    report: true
    out: Results/FCclasses/rate_report.txt
    json: Results/FCclasses/rate_report.json
    tsv: Results/FCclasses/rate_report.tsv
    print: true
  db:
    rebuild: true
  obsidian:
    export: true
```

规则如下：

- **finalize** / **finalize.enabled** 控制总开关；默认值为 **true**。未显式配置 **finalize.actions** 或各类快捷开关时，默认动作集为 **method.generate**、**db.rebuild** 与 **obsidian.export**。写 **finalize: false** 或 **finalize.enabled: false** 可关闭隐式收尾；写 **finalize.method.generate: false** 可只从默认动作集中去掉方法学生成，保留数据库重建与 Obsidian 导出。只要显式提供了任一 **finalize.actions**、**finalize.method.generate**、**finalize.fcclasses.report**、**finalize.db.rebuild** 或 **finalize.obsidian.export**，系统也会自动视为已启用。
- **finalize.scope** 只接受 **case** 或 **project**。**case** 作用于当前 case root；**project** 会先回溯 project root，再对该根目录执行对应动作。
- **finalize.actions** 接受 YAML sequence 或逗号分隔字符串；动作包括 **method.generate**、**fcclasses.report**、**db.rebuild** 与 **obsidian.export**。别名 **method.export**、**btproc method.generate**、**fcclasses.rates.report**、**database.rebuild**、**btdb.rebuild** 和 **btdb.obsidian.export** 会归一到相应动作。
- **method.generate** 会调用 **btproc method generate --scope <case|project> --path <root> --task-file <file> --origin <origin>**，默认输出 **Results/Method/method.md** 与 **Results/Method/method.json**。
- **finalize.method.out**、**finalize.method.json**、**finalize.method.refs**、**finalize.method.strict** 可作为快捷配置；即使没有显式写 **finalize.method.generate: true**，这些选项也会作用于默认的 **method.generate**。在 **actions:** 中也可写为 **method.generate** 的 **out**、**json**、**refs**、**strict** 选项。
- **finalize.fcclasses.report** 控制 FCclasses 汇总；当前任务文件存在有效 FCclasses 任务时默认启用。写为 **false** 可保留其他收尾动作但关闭该汇总。
- **finalize.fcclasses.out**、**finalize.fcclasses.json**、**finalize.fcclasses.tsv** 用于覆写三种汇总文件路径；默认分别为 **Results/FCclasses/rate_report.txt**、**rate_report.json** 和 **rate_report.tsv**。**finalize.fcclasses.print** 控制是否把文本汇总打印到收尾任务日志，默认值为 **true**。

- **finalize.db.rebuild: true**、**finalize.obsidian.export: true** 可作为快捷布尔开关，与 **finalize.actions** 共同合并去重。
- **finalize.obsidian.out** 用于覆写 Obsidian 导出目录；未显式给出时，默认输出到作用域根目录下的 **Results/Obsidian**。
- **finalize.task_name** 用于自定义隐式收尾任务名；默认任务名为 **__banetask_finalize__**，对应目录为 **<case_root>/__banetask_finalize__/**。
- 系统生成的隐式 finalize 任务默认带 **localfirst: true**。在 scheduler allocation 内触发 autorun 时，若当前节点资源可立即取得，收尾脚本会直接 local 执行；否则仍按通常的 target / queue 规则提交。
- 只有当当前文件中所有仍然有效的任务分支都已完成时，该隐式任务才会被物化；被 **%when** 判定为不生效的分支会从收尾阻塞集合中排除。
- 在普通完整模式中，Derive pipeline 与 Plot pipeline 会先于隐式收尾任务运行。因此收尾脚本可面向同轮生成的 **Results/Derived/**、**Results/Plots/** 与对应 artifact 执行数据库重建、导出或方法学说明生成。

  当计划重新变为未完成状态，或显式关闭 **finalize** 时，系统会自动清理已生成的隐式收尾目录、workflow 与 **bt.status** 记录。

2.1.2.6 define

define: 用于定义可在后续任务描述中重复使用的文本占位符。它本质上是一层轻量的参数化机制，适合把方法、基组、溶剂名、标签或路径等重复出现的内容集中声明，再通过 **[name]** 的形式在任务块中引用。例如：

```
define:
  method: B3LYP
  basis: 6-31G(d)
```

调用形式如下：

```
%keywords "opt [method]/[basis]"
```

占位符替换适用范围包括：

- 任务名
- **%source**
- **%keywords**
- **%extrainput**
- **%preprocess** / **%process**
- **%when**
- **%meta**
- **%plot**

define: 的条目必须以至少两个字面空格缩进，并在第一个冒号处分成键和值。值最外层的一对单引号或双引号会被去掉。空行和纯注释行可出现在块内，它们会被忽略，不会终止 **define:**；下一条非空、非注释且未按条目缩进的内容才结束该块。

```
define:
  method: B3LYP

  # basis set
```

```
  basis: 6-31G(d)
```

`define:` 是轻量键值语法，不支持把缩进子映射、序列或多行 YAML 值当作一个占位符值。需要结构化集合时使用普通 YAML、`matrix:` 或命名集合。

2.1.2.7 matrix

`matrix:` 用于把一组逻辑任务自动展开为多组参数变体，是 BaneTask 进行批量组合计算的核心机制之一。每个键都可以声明一个逗号分隔的取值列表，解析器随后会对这些维度做笛卡尔积展开；若单个值本身包含逗号，则应使用单引号或双引号将该值包围。

```
matrix:
  method: B3LYP, CAM-B3LYP
  solvent: dmso, h2o, dichloromethane
```

除最常见的 `key: v1, v2, ...` 写法外，也支持扩展列表语法：

```
matrix:
  - method: B3LYP, CAM-B3LYP, wB97XD
    name: b3lyp, cam, wb97xd
  - basis: def2-SVP, def2-TZVP, def2-TZVPP
    map: method
```

扩展字段含义如下：

- `name:` 为当前轴的每个值显式指定任务名后缀 token；它只影响展开后的任务名显示，不改变 `[method]` 等变量替换时看到的真实值。
- `map:` 将当前轴与另一条独立轴做一一对应绑定，形成配对展开关系；适合表达“方法 A 对应基组 A，方法 B 对应基组 B”这类组合。
- 使用 `map:` 时，当前轴与目标轴的 value 个数必须一致；`map` 目标必须存在，且不能形成链式映射。
- 一个 `matrix:` 块内不能混用简单 mapping 语法与扩展 list 语法。

展开行为如下：

- 每个任务块都会派生出多个真实任务名。
- 真实任务名后缀为 `__key_value__key2_value2`。
- 任务元数据中会写入：
  - `template_task_name`
  - `matrix_variant_key`
  - `matrix.<name>`
- 对任务名的引用会自动改写到对应变体，包括：
  - `%source <task>`
  - `%control guess <task>`
  - `%when completed <task>`
  - `%when {{ task.field }}`
  - `%plot` 表达式中的任务引用、`from task` 与任务级默认输出占位符

当 matrix 值中包含空格、括号或其他非字母数字字符时，任务名后缀中的显示值会做安全化处理；数据库元数据中的 `matrix.<name>` 保留原始值。

2.1.2.8 结构化 YAML 集合

除 `define` 与 `matrix` 之外，YAML 头部还可以声明结构化集合。集合本身只是头部数据，通常用于被后续 `@foreach` / `@foreach?` 预展开指令消费，或由工具写入 `extra.yaml` 后在下一次刷新时参与任务展开。

```yaml
frag:
  - name: donor
    select: 1-4
    charge: 0
    spin: 1
    totfrag: 2
    totatom: 4
  - name: bridge
    select: 5-8
    charge: 0
    spin: 1
    totfrag: 2
    totatom: 4
```

规则如下：

- 集合键对应一个 YAML sequence。
- 序列元素可以是标量、对象或嵌套对象；嵌套对象在展开时会被扁平化为点路径，例如 `state.index`。
- 若序列元素本身是标量，后续展开时可通过 `[collection.value]` 访问该值。
- 由几何解析自动写入的 `frag:` 集合会包含片段名、原子选择、电荷、自旋等字段，并在展开时可继续提供总片段数与当前片段原子数。

2.1.3 &INCLUDE

`&INCLUDE` 用于把另一个文件、匿名片段或命名片段并入当前位置。目标会按当前位置所期待的 DSL 类型解析后插入，而不是先做无条件的全文件文本替换。它适合复用 YAML 变量、任务块、控制块、前后处理命令和 `.btlib` 片段库。

`&INCLUDE` 有四种常用形式：

```
&INCLUDE "common.yaml"                        # 并入文件
&INCLUDE "lib/analysis.btlib" as analysis # 导入库并绑定命名空间
&INCLUDE <analysis.btlib> as analysis     # 从 BANETASK_BTLIB_PATH /
↪  ~/.bane/task/btlib 查找库文件
&INCLUDE analysis.hole_ele(dir="excit")   # 展开命名片段
```

双引号文件路径始终相对于写出该 `&INCLUDE` 的源文件解析；因此嵌套片段和库依赖也各自以其所在文件为基准。尖括号路径 `&INCLUDE <...>` 从 `BANETASK_BTLIB_PATH` 指定的库目录查找，适合共享库文件。建议相对文件目标加引号，共享库目标用尖括号，库符号保持不加引号。

2.1.3.1 顶层行为

文件顶层支持文档片段与库导入：

- `.yaml` / `.yml`：合并普通元数据、`define`、`matrix` 和结构化集合。

- **.bt / .projbt**：并入完整任务文档，追加其中的任务块，并继承其文件级元数据。包含文件中已显式声明的同名元数据优先于被包含文档；多个被包含文档提供同名元数据时，先并入的值保留。
- **.btfrag**：按 **document** 片段解析，可在当前位置展开一组完整任务块。
- **&INCLUDE "file.btlib" as ns**：加载命名片段库；**.btlib** 不带 **as** 时视为错误。
- **&INCLUDE <file.btlib> as ns**：从 **BANETASK_BTLIB_PATH** / **~/.bane/task/btlib** 加载命名片段库；同样必须带 **as**。
- 父文件在顶层导入的库命名空间，也可被随后并入的 **.bt** / **.projbt** 任务使用；子文件自行导入的命名空间只在该子文档及其后代中有效，不会泄漏到父文件或兄弟文件。

例如：

```
&INCLUDE <analysis.btlib> as analysis
&INCLUDE "tasks/common.bt"
&INCLUDE "tasks/optional-workflow.btfrag"

$td
  %process
    &INCLUDE analysis.hole_ele
```

2.1.3.2 片段类型

任务内部的 **&INCLUDE** 按所在上下文解释：

&INCLUDE 所在位置	期待的片段类型	典型内容
文件顶层	**document**	**$opt**、**$td** 等完整任务块
任务体内、各 **%...** 块之外	**task_body**	**%control**、**%process**、**%when**、**%plot** 等任务指令
%process	**process**	后处理命令列表
%preprocess	**preprocess**	前处理命令列表
%control	**control**	**totcore**、**totmem** 等控制项
%mod	**mod**	**charge**、**spin**、**geom**、**basisset** 等修改项
%when	**when**	条件列表
%meta	**meta**	任务元数据项
%param	**param**	模板参数正文
%args	**args**	命令行参数片段
多行 **%batch**	**batch**	**lane**、**collect**、**cluster** 等规则
空的 **%body** / **%extrainput** 块	**text**	原样脚本或程序输入文本

例如，空穴—电子分析片段可以写成：

```
$td
  %source opt
```

```
  %control
    totcore 4
    totmem 4000
  %keywords "td [func]/[bset] [scrf] [DFT-D]"
  %process
    &INCLUDE "hole.part.bt"
```

其中**hole.part.bt**:

```
banewfn hole-ele [inputname].fchk
banewfn fmo [inputname].fchk
cd excit && baneov ../[inputname].log
cd ..
publish --pin excit
```

这里外部文件按**process**命令列表解析;空行与注释只用于排版,不会提前结束**%process**。

任务体片段可以一次并入多种任务指令:

```
$td
  &INCLUDE "mixins/normal-task.btfrag"
  %control
    totcore 8
```

mixins/normal-task.btfrag:

```
&FRAGMENT task_body

%control
  totcore 4
  totmem 4000

%process
  publish .
```

片段在**&INCLUDE**所在位置展开，之后出现的配置继续遵循原有 DSL 合并规则；例如上例最终使用**totcore 8**，同时保留**totmem 4000**。

2.1.3.3 独立片段文件

普通文件可直接作为匿名片段使用，片段类型由当前位置推断。**.btfrag**还可在首部写一个可选的类型声明:

```
&FRAGMENT process

banewfn fmo [inputname].fchk
publish --pin excit
```

独立片段文件只能包含一个匿名片段，且不接受调用参数；需要参数化或在一个文件中定义多个片段时，应使用**.btlib**。

2.1.3.4 定义和调用 .btlib

库文件由一个或多个命名**&FRAGMENT**组成:

```
&FRAGMENT control normal(ncore=4, mem=4000)
totcore [ncore]
totmem [mem]

&FRAGMENT process hole_ele(
  dir="excit",
  fchk="[inputname].fchk",
  log="../[inputname].log"
)
banewfn hole-ele [fchk]
banewfn fmo [fchk]
cd [dir] && baneov [log]
publish --pin [dir]
```

使用时仍只需要 `&INCLUDE`：

```
&INCLUDE <analysis.btlib> as analysis

$td
  %control
    &INCLUDE analysis.normal(ncore=8)
  %process
    &INCLUDE analysis.hole_ele(dir="hole")
```

片段类型为：

```
document  task_body  process  preprocess  control  mod
when      meta       param    args        batch    text
```

其中`task` 可作为 `task_body` 的简写，`doc` 可作为 `document` 的简写，`body` / `extrainput` 可作为 `text` 的别称；`command`、`commands` 和 `cmd` 可作为 `process` 的别称。由于 `%process` 与 `%preprocess` 使用同一种命令列表语法，`process` / `preprocess` / `command` 片段可以在这两个上下文之间复用。其他片段类型与调用位置不兼容时，解析器会在调用位置报错，而不会把内容误当成其他 DSL。

2.1.3.5 参数规则

参数继续使用 BaneTask 约定的 `[name]` 占位形式：

```
&FRAGMENT process report(dir, file="[inputname].log")
cd [dir]
grep "SCF Done" [file]
```

调用参数必须使用命名形式：

```
&INCLUDE analysis.report(dir="results")
```

规则如下：

- 无默认值的参数为必填参数；缺失时立即报错。
- 调用值覆盖默认值；未声明的调用参数会报错。
- 默认值和调用值可以引用同一片段中的其他参数，解析不依赖参数声明顺序；参数引用形

成循环时会显示引用链并报错。

- 片段参数先替换；`[inputname]`、`[func]` 等普通任务或全局变量仍留给后续正常变量展开阶段处理。

2.1.3.6 库依赖与片段嵌套

库可在第一个 `&FRAGMENT` 之前导入依赖：

```
&INCLUDE "base.btlib" as base

&FRAGMENT process wrapped(value="default")
echo begin-[value]
&INCLUDE base.tail(value="[value]")
```

库内解析采用词法优先级：库自身声明的依赖命名空间优先于调用文件中同名的顶层命名空间，因此调用环境不会意外改变库的依赖绑定。

同一库内的兄弟片段可以直接按名称引用；`self.` / `this.` 前缀仍作为显式写法保留：

```
&FRAGMENT command pair(prefix="item")
echo [prefix]-first
&INCLUDE tail(prefix="[prefix]")

&FRAGMENT process tail(prefix)
echo [prefix]-second
```

在库外调用命名片段仍须使用 `namespace.name`，从而避免与文件路径混淆。

库导入循环、独立片段循环和片段调用递归都会显示完整 include 链并终止解析。

2.1.3.7 文本片段与 heredoc

空的 `%body` / `%extrainput` 可开启文本块，并通过相同语法并入原始文本：

```
$script
  %program script
  %body
    &INCLUDE "scripts/run.sh"
  %extrainput
    &INCLUDE "inputs/common.inp"
```

`text` 片段是不可解释的正文：shebang、`#` 注释、`$HOME`、程序原生 `%...` 行和字面量 `&INCLUDE` 都会保留；只执行片段参数替换。heredoc 内部同样始终是字面量，因此下面的 `&INCLUDE` 不会展开：

```
%body << EOF
&INCLUDE "literal-text.sh"
EOF
```

2.1.3.8 组合与错误原则

`&INCLUDE` 在当前位置按结构展开：命令、条件等顺序型内容按出现顺序追加；键值或单值型指令继续服从原有“后出现者覆盖”的任务解析规则。解析器会拒绝以下情况：

- 片段类型与调用上下文不匹配；

- 在任务块内部导入库命名空间；
- `.btlib` 未使用 `as <namespace>`；
- 必填参数缺失、未知参数或重复参数；
- 命名空间冲突、未知片段或任何形式的循环 include。

这套机制保留了既有顶层 YAML、`.bt` 和 `.projbt` include 行为，同时让相同的 `&INCLUDE` 文本形式覆盖文件拆分、结构片段和参数化库复用。需要检查最终展开结果时，可运行 `btool expand task.bt`。

2.1.3.9 自动并入文件

除显式 `&INCLUDE` 外，解析器还会自动尝试并入与当前任务文件同目录下的若干约定文件。这一机制主要用于放置局部共享变量或目录级公共任务块，从而减少重复书写。

- `extra.yaml`
- `extra.bt`

这两个文件通常用于局部补充变量、矩阵或共享任务块。其中 `extra.yaml` 也常作为 `bt-proc fork collect` 的默认输出，用来为 `@foreach?` 提供可选集合。

2.1.4 `@foreach` / `@foreach?` 文本预展开

`@foreach` / `@foreach?` / `@end` 用于在任务块进入 DSL 解析前执行文本级预展开。它适合批量生成一组只在少量参数上不同的任务块、脚本任务或 `%mod` 片段。典型写法如下：

```
frag:
  - name: donor
    select: 1-4
    charge: 0
    spin: 1
  - name: bridge
    select: 5-8
    charge: 0
    spin: 1

@foreach frag
$[frag.name]
  %source opt
  %mod
    geom select [frag.select]
    charge [frag.charge]
    spin [frag.spin]
    note [frag.index]
  %keywords "nosymm [func-E]/[bset-E] [scrf-E] [DFT-D]"
@end
```

规则如下：

- `@foreach <collection>` 消费 YAML 头部或自动并入文件中的同名集合。
- 循环体在进入 DSL 解析前完成预展开，因此 `$task` 名、`%mod`、`%extrainput`、`%preprocess`、`%process` 等位置都可以引用展开变量。
- 字段引用统一写成 `[集合名.字段名]`，例如 `[frag.name]`、`[frag.select]`；若序列元

素本身是标量，则使用 `[frag.value]`。

- 解析器会为每个集合元素自动补一个从 1 开始的 `[frag.index]`，因此即使原 YAML 未写 `index:`，也可以直接引用。
- 对由几何解析自动生成的 `frag:` 集合，系统还会补充 `[frag.totfrag]`（总 fragment 数）和 `[frag.totatom]`（当前 fragment 的原子数）。
- `@foreach` 支持嵌套；若集合未定义或缺少 `@end`，解析器会直接报错。
- `@foreach? <collection>` 表示可选集合：若集合当前不存在，则整个循环体会被直接跳过，不报错。该形式适合与`fork`配合。

`fork` 可在后处理阶段把结构集合写入自动并入文件；下一次解析时，`@foreach?` 只在集合存在时展开。完整组合示例见 `recipes/foreach-fork.md`。

也可以在声明处构造数值序列：

```
@foreach item in series(10-15)
$calc_[item.series]
  %source origin
  %keywords "sp b3lyp/6-31g*"
  %preprocess
    echo idx=[item.index] value=[item.series]
@end
```

说明：

- `series(10-15)` 会按闭区间展开为 `10, 11, 12, 13, 14, 15`；若写成 `series(5-3)`，则会按 `5, 4, 3` 逆序展开。
- `[item.series]` 与 `[item.value]` 都表示当前数值。
- `[item.index]` 为自动生成的 1 基序号。
- `@foreach? item in series(3-5)` 与普通 `@foreach item in series(...)` 一样会实际展开，只是保留了“可选循环”外壳，便于与统一模板写法保持一致。
- `@foreach frag` / `@foreach? frag` 也会被解析。

2.1.5 别名与替代写法

任务文件应优先使用下列规范写法；解析器同时接受列出的替代写法：

- `%source <task> guess`：可读取的输入别名；规范形式为 `%source <task>` 与 `%control guess true|<task>`。
- `%program other` 下的 `%keywords "job=..."`：作为输入别名；规范写法为 `%template` + `%param`，详见other。
- `%program script` 下的 `%keywords`（runner）：作为输入别名；规范写法为 `%runner`，详见script。
- `%control runif` / `runifdef`：不是有效指令；条件控制应使用独立 `%when`。
- 批量规范化任务文件时，`btool lint` 可报告非规范输入，`btool migrate` 可输出 canonical DSL v2。

2.2 任务块指令

任务块指令决定任务的几何来源、程序类型、资源约束、输入内容、执行条件、前后处理和任务元数据。

2.2.1 指令总览

任务块中的各类指令分别用于确定来源、指定程序、约束资源、控制生成、补充输入和补充元数据。实际使用时，这些指令会共同决定输入内容、依赖关系和 workflow 行为。

支持以下任务块指令：

- **%source**：为常规程序声明主几何、命名辅助几何与默认上游依赖；为 FCclasses 声明结构化电子态和耦合资源。
- **%program**：指定程序类型。
- **%control**：指定资源、guess、最终几何文件和若干生成/完成判定开关。
- **%interface**：声明 Gaussian External 接口提供者或消费者。
- **%when**：指定任务生成条件。
- **%keywords**：程序相关的主输入字段；具体语义见第 3 章对应程序。
- **%extrainput**：程序相关的附加输入字段；具体语义见第 3 章对应程序。
- **%template**：指定 **%program other** 的模板名。
- **%param**：指定 **%program other** 的 **key=value** 参数。
- **%runner**：指定 **%program script** 的 runner / shebang。
- **%body**：指定 **%program script** 的脚本正文。
- **%batch**：指定多帧 XYZ 批处理的并发 lane 数、产物收集和可选构象聚类；适用于 script、xtb、gaussian、orca、gamess、amesp、mopac、bdf、mrcc、maple、other。
- **%mod**：修改生成输入中的电荷、自旋、输出坐标片段和受支持程序的结构化基组等输入。
- **%preprocess**：指定主程序运行前命令。
- **%process**：指定主程序运行后命令。
- **%meta**：指定任务级结构标识。
- **%plot**：声明与当前任务关联的绘图 artifact；默认以 **self** 为结果上下文，不生成计算任务目录。

2.2.2 %source

BaneTask 的任务控制文件通常不直接包含几何；运行时需要指定几何来源来执行预定任务。**%source** 用于声明当前任务应从哪里取得几何，并在必要时隐式建立对上游任务的默认依赖。该指令既影响输入文件中的坐标来源，也影响部分 **%when** 条件在省略任务名前缀时的默认解释方式。

2.2.2.1 原始输入文件查找顺序

给定 case 入口文件 **caseA.bt**，BaneTask 会按照以下顺序在 **caseA.bt**的同目录查找原始几何输入：

1. **caseA.xyz**
2. **caseA.gjf**(Gaussian 输入)
3. **caseA.com**(Gaussian 输入)
4. **caseA.inp**（ORCA / GAMESS / MAPLE 输入，按文件内容识别）
5. **caseA.aip**（AMESP 输入，按文件内容识别）
6. **caseA.log**（Gaussian / ORCA / GAMESS / MAPLE 输出日志，按文件内容识别）
7. **caseA.aop**（AMESP 输出日志，按文件内容识别）
8. **caseA.out**（ORCA / MAPLE / MOPAC 等输出日志，按文件内容识别）
9. **caseA.gms**（GAMESS 输入或输出，按文件内容识别）

若入口文件同目录没有找到来源文件，系统还会在当前工作目录按同一顺序查找。

来源文件分为三类处理：程序输出日志读取最终几何；程序输入文件读取显式坐标；XYZ 读取指定帧或最后一帧。XYZ 第二行中的 key-value metadata 可写入 **normalize.yaml** / **extra.yaml** 的 **define:** 区域；程序输出日志中的程序名、版本、method、termination 等信息作为解析元数据保留，不写入模板 define。

特别地，**.gjf** 与 **.com** 可以包含 fragment 信息。BaneTask 会尝试按照 Gaussian 输入格式读取坐标、电荷、自旋以及 fragment 标签。若解析结果包含 fragment 集合，运行期还会在 case 根目录的 **extra.yaml** 中自动写入或更新顶层 **frag:** 集合（带 **# BEGIN BANETASK AUTO FRAG** / **# END BANETASK AUTO FRAG** 标记），每个片段项会包含 **name**、**select**、**charge**、**spin**、**index**，以及自动补充的 **totfrag**（总 fragment 数）和 **totatom**（当前 fragment 原子数）。

2.2.2.2 %source 的其他取值

```
%source origin
%source restart
%source xyz=seed_geom
%source xyz=seed_geom frame=2
%source opt
```

各取值语义如下：

- **origin**：使用 case 原始输入文件。省略 **%source** 时等同于 **origin**。
- **restart**：使用当前任务目录中已存在日志文件中的几何，并在成功解析后将已有工件移入 **fail/** 目录。
- **xyz=<name>**：使用 case 根目录下的 **<name>.xyz**。
- **xyz=<name> frame=<N>**：对多帧 XYZ 文件，取第 **N** 帧作为几何来源；**N** 为 1-based 正整数。
- **<task>**：使用上游任务 **<task>** 的主输出文件几何；该任务同时成为默认依赖。

其中，**restart** 最常用于任务失败后的续算，或者在整个项目更换计算级别但希望沿用当前任务目录中最新结构时使用。

当显式 XYZ 源写成 **xyz=<name> frame=<N>** 时，BaneTask 会把 **<name>.xyz** 视为多帧 XYZ，并选取第 **N** 帧（从 1 开始计数）作为当前任务的几何来源。

2.2.2.3 上游任务作为几何来源时的规则

当 **%source <task>** 指向上游任务时：

- 当 **%source** 指向上游任务时，该任务自动进入当前任务依赖列表。
- 若上游任务声明了 **%control finalgeom <xyz-file>**，则下游优先读取上游任务目录中的 **<xyz-file>**，并把该文件存在性作为上游完成判据；此时不会再要求上游具备常规程序日志或日志正常结束标志。
- 若上游任务没有声明 **finalgeom**，几何来源文件优先读取上游任务目录中的 **task.manifest.json** / workflow 所记录的主输出文件；若缺少这些 sidecar，则回退到 **<task>/<task>_<origin>.log**。
- **%when** 中引用上游结果时，默认依赖任务即为 **%source** 指定的任务。

2.2.2.4 命名几何来源

一个任务需要同时消费多个结构时，可把 `%source` 写成多行命名绑定。`geom` 表示主几何，`geom.<slot>` 表示辅助几何：

```
$compare
  %program orca
  %source
    geom Raw
    geom.reference Product
  %keywords "NEB-TS PBEh-3c"
  %extrainput << EOF
%neb
 NEB_END_XYZFILE [geom.reference] # 末态结构
end
EOF
```

每行格式为：

```
geom[.<slot>] <source-ref> [option=value ...]
```

`<source-ref>` 与单行 `%source` 使用同一套来源语义，可取 `origin`、`restart`、`xyz=<name>`、`xyz=<name> frame=<N>` 或上游任务名。内建来源名只按规范小写识别，因此名为 `Origin` 或 `Restart` 的任务仍可作为普通任务引用。

单行写法与主几何绑定等价：

```
%source Raw
```

等价于：

```
%source
  geom Raw
```

命名绑定遵循以下规则：

- 存在任意辅助几何时必须同时声明 `geom`。`geom.main` 可作为输入别名，但格式化输出统一写成 `geom`。
- `<slot>` 必须以字母开头，只能包含字母、数字、下划线和连字符；解析后统一使用小写名称。
- 所有指向任务的绑定都会自动加入依赖列表，并在 matrix、`@foreach` 和任务重命名时同步改写。
- 每个几何独立解析坐标、电荷和自旋。不带作用域的 `%mod charge`、`%mod spin` 和 `%mod geom` 只作用于主几何 `geom`，辅助几何不会隐式继承主几何的修改。
- 可用 `charge.<slot>`、`spin.<slot>` 与 `geom.<slot> select/remove` 显式修改命名几何；目标 slot 必须已由 `%source` 声明。`.main` 可作为输入别名，但格式化输出会恢复为无作用域主几何写法。
- `%mod` 不提供 `charge2`、`geom2` 等位置目标；数字形式只保留给运行期占位符。`basisset`、`freeze`、`oniom`、`fragment` 等程序输入操作仍采用既有的主几何语义。
- 核心运行时只负责解析、修改、物化和暴露命名几何，不根据 slot 名猜测 NEB、QST2、复合物或嵌入电荷等程序语义。具体如何消费辅助来源由任务正文、模板或程序后端决定。

例如，可在不改变主几何的情况下单独调整参考结构：

```
$compare
  %program script
  %source
    geom Raw
    geom.reference Product
  %mod
    charge.reference 0
    spin.reference 1
    geom.reference select 1-20
  %runner bash
  %body << EOF
printf '%s\n' '[charge.reference] [spin.reference]'
cp '[xyz_file.reference]' reference.xyz
EOF
```

解析并应用各自 `%mod` 后，几何会在任务准备期间先物化为标准 XYZ：

```
<task>_<origin>.xyz
<task>_<origin>.<slot>.xyz
```

辅助几何 `<task>_<origin>.<slot>.xyz` 会保留，供输入、命令或后续 `comd-only` 刷新使用。主几何的同 stem 文件 `<task>_<origin>.xyz` 属于准备阶段产物；当它不是实际程序输入、没有被生成的输入文件、workflow 或命令脚本引用，也不是 `restart` 或 source 占位符恢复所需快照时，BaneTask 会在任务产物生成完成后自动删除。XTB 以该 XYZ 作为实际输入，因此不会被清理；显式使用 `[xyz_file]` 的任务同样会保留该文件。

主几何继续使用现有占位符，并同时提供显式 `.main` 别名。辅助几何使用命名 slot：

内容	主几何	主几何显式别名	辅助几何示例
坐标正文	`[geom]`	`[geom.main]`	`[geom.reference]`
当前任务目录中的 XYZ 文件名	`[xyz_file]`	`[xyz_file.main]`	`[xyz_file.reference]`
电荷	`[charge]`	`[charge.main]`	`[charge.reference]`
自旋多重度	`[spin]`	`[spin.main]`	`[spin.reference]`
未成对电子数	`[uhf]`	`[uhf.main]`	`[uhf.reference]`
来源描述	`[source]`	`[source.main]`	`[source.reference]`
原子数	`[atom_count]`	`[atom_count.main]`	`[atom_count.reference]`

`[geom.<slot>]` 只包含坐标行，不包含 XYZ 的原子数和注释行；`[xyz_file.<slot>]` 是当前任务目录内的文件名，不直接引用上游任务目录。`[charge.<slot>]`、`[spin.<slot>]`、`[uhf.<slot>]`、`[geom.<slot>]` 与 `[atom_count.<slot>]` 均反映该几何应用 scoped `%mod` 后的最终值，其中 `[uhf.<slot>]` 等于对应自旋多重度减一。

位置式模板还可以使用第二个及后续几何还提供 `[geom2]`、`[xyz_file2]`、`[charge2]`、`[spin2]`、`[uhf2]` 等数字别名；数字按几何绑定的声明顺序分配，主几何固定为第一个。正式

任务文件应优先使用命名 slot，避免调整绑定顺序后改变含义。

这些占位符可用于输入关键词和附加输入、XTB 参数、script 正文、**other** 模板及参数、前后处理命令等运行期文本。**geom**、**xyz_file**、**charge**、**spin**、**uhf**、**source**、**atom_count** 及其命名或数字变体属于系统保留名，不能由 **define:**、matrix 轴或 **%param** 覆盖。引用不存在的 slot 或拼错系统占位符会在任务准备阶段直接报错。

task.manifest.json 会记录每个来源声明、解析路径、电荷、自旋以及该 binding 实际应用的规范化 **%mod** 列表；只有仍保留的物化文件才写入 **staged_file**。**comd-only** 刷新命令脚本时只会在来源声明和对应 **%mod** 都未变化时恢复已经物化的命名来源；普通来源缺少 staged 文件时会按声明重新解析，**restart** 来源则保留其快照，缺少可用清单时需要先正常准备任务一次。

2.2.2.5 FCclasses 结构化资源来源

%program fcclasses 不以单个主几何为输入，而是直接绑定上游电子态、跃迁矩和耦合资源。这类任务应使用多行 **%source**：

```
$isc
  %program fcclasses
  %source
    state1 td state=S1
    state2 ut3 state=T3
    soc soc-tddft pair=S1,T3 component=prefer-ms
  %keywords "isc(both)"
```

每行格式为：

```
<role> <task-name> [selector=value ...]
```

支持的角色如下：

角色	数量	用途
state1	必须且只能有一个	当前跃迁的初态
state2	至少一个，可重复	当前跃迁的末态；每个 **state2** 分别形成一个态对
soc	可选	SOC 输出或可解析的 SOC 资源
nac	可选	非绝热耦合文件、BDF NAC 输出或可供 **gen_fcc_dipfile** 转换的数据
eldip	可选	电跃迁偶极文件或可供 **gen_fcc_dipfile** 转换的数据
magdip	可选	磁跃迁偶极文件，或可供 **gen_fcc_dipfile** 转换的数据
nrcoup	可选	FCclasses scalar coupling 文件
asset	可选，可重复	需要按指定名称复制到每个作业目录的普通文件

支持的选择器如下：

选择器	作用	示例
`state=`	显式指定态标签；当前可识别 `S0`、`S1`、`T1` 等 singlet/triplet root	`state=T3`
`pair=`	将偶极、NAC、SOC 或 scalar coupling 绑定到特定态对	`pair=S1,T3`
`file=`	精确指定上游任务目录内的安全相对路径	`file=result.fcc`
`artifact=`	按上游 manifest/任务目录中的文件路径、basename 或 stem 缩小候选范围	`artifact=td.fchk`
`component=`	SOC 分量策略：`prefer-ms`、`prefer-xyz`、`ms-only`、`xyz-only`	`component=prefer-xyz`
`variant=`	BDF NAC 版本：默认 `corrected`，也可取 `raw`	`variant=raw`
`as=`	`asset` 在作业目录中的安全相对文件名	`as=custom.dat`

结构化 `%source` 中引用的任务都会自动进入依赖列表；任务重命名、matrix 和 `@foreach` 展开也会同步改写这些任务名和选择器中的变量。多个候选文件无法唯一确定时，输入生成会列出歧义并要求补充 `file=` 或 `artifact=`，不会静默选择目录中的第一个文件。

重复 `state2` 可用于一次声明多个末态：

```
%source
  state1 td state=S1
  state2 ut1 state=T1
  state2 ut2 state=T2
  state2 ut3 state=T3
  soc soc-tddft
```

该任务分别规划 `S1` 与 `T1`、`T2`、`T3` 的作业。若同一角色为不同态对提供不同文件，应在对应绑定上添加 `pair=`，使每个态对只匹配一个资源。

2.2.2.6 `restart` 的工件整理

`%source restart` 解析成功后，会将当前任务目录中以下已有工件移入 `fail/`、`fail1/`、`fail2/` 等目录：

- `.gjf`
- `.chk`
- `.log`
- `.inp`
- `.xyz`

该行为用于保留重启前的任务快照，并为新一轮输入和日志留出干净目录状态。

2.2.3 `%program`

`%program` 用于声明当前任务应采用哪一类输入生成器。它不仅决定要生成的文件类型，也决定 `%keywords`、`%extrainput`、`%template`、`%param`、`%runner`、`%body`、`%batch` 与 `%control guess` 的解释方式。

```
%program gaussian
%program orca
%program gamess
%program amesp
%program mopac
%program bdf
%program fcclasses
%program mrcc
%program maple
%program xtb
%program script
%program other
```

程序类型说明如下：

类型	生成物	正式输入字段
`gaussian`	Gaussian 输入文件 `*.gjf`	`%keywords` + 可选 `%extrainput`
`orca`	ORCA 输入文件 `*.inp`	`%keywords` + 可选 `%extrainput`
`gamess`	GAMESS 输入文件 `*.inp`	`%keywords` + 可选 `%extrainput`
`amesp`	AMESP 输入文件 `*.aip`	`%keywords` + 可选 `%extrainput`
`mopac`	MOPAC 输入文件 `*.mop`	`%keywords` + 可选 `%extrainput`
`bdf`	BDF 输入文件 `*.inp`	`%keywords` + 可选 `%extrainput`
`fcclasses`	FCclasses 计划、独立作业目录与 `*.fcplan` 启动器	结构化 `%source` + 可选 `%keywords` / `%extrainput`
`mrcc`	MRCC 固定输入文件 `MINP`	`%keywords` + 可选 `%extrainput`
`maple`	MAPLE 输入文件 `*.inp`	`%keywords` + 可选 `%extrainput`
`xtb`	标准化 `*.xyz` 与可选 `*.xcontrol` sidecar	`%args` + 可选 `%extrainput`
`script`	POSIX `*.sh` 或 Windows `*.bat`	`%runner` + `%body`
`other`	由模板生成的任意程序输入	`%template` + 可选 `%param`

别名与自动推断：

- `%program other` 另接受 `%keywords "job=... key=value"`；解析器会接受该写法，`btool lint` 会报告其规范替代形式。
- `%program script` 另接受用 `%keywords` 指定 runner；应使用 `%runner`。
- 未显式指定 `%program` 时，默认程序类型仍为 `gaussian`；若任务只提供 `%args`、且没有 Gaussian / ORCA / GAMESS / AMESP / MOPAC / BDF / FCclasses / MRCC / MAPLE / other / script 的输入描述，解析器会把该任务判定为 `%program xtb` 并给出提示。

2.2.4 `%control`

`%control` 是一个多行块，用于描述当前任务在生成阶段所需的资源和执行控制项。程序关键字继续写在 `%keywords`、`%args`、`%template` / `%param` 或 `%body` 等输入字段中；`%control`

写核心数、内存、guess 行为以及执行判定相关开关。支持以下设置：

```
%control
  totcore 32
  totmem 96000
  gpus 1
  gpu_type A100
  gpu_mem 40000
  guess true
  waitfreq false
  alert true
  localfirst true
  externalinp true
  verbosity p
  geom check
  finalgeom alllast.xyz
```

其中字符串型控制值会在运行时展开通用任务占位符，例如 **[rootname]**、**[taskname]**、**[inputname]**、**[originname]** 与 **[sourcename]**。这适用于 **guess <task>**、**finalgeom <xyz-file>** 以及 GPU 类型字段 **gpu_type**；例如 **finalgeom [inputname].xyz** 会在任务 **opt** 处理源结构 **mol.xyz** 时解析为 **opt_mol.xyz**。

2.2.4.1 totcore <N>

totcore 用于指定当前任务的总核心数。该值会同时影响输入生成阶段和 workflow 头部中的 **nprocs** 设置；MOPAC 输入会把它写为 **THREADS=<N>**，并覆盖 **%keywords** 中的 **THREADS**；对于 **other** 类型任务，还会进一步覆写模板参数中的 **nprocs**，从而保证模板输入与实际调度资源保持一致。

2.2.4.2 totmem <MB>

totmem 用于指定当前任务的总内存，单位为 MB。对于 Gaussian，系统会直接把该值写入对应的内存指令；对于 ORCA，系统会根据 **totcore** 自动换算 **maxcore = totmem / totcore**，若未显式指定 **totcore**，则按总内存进行处理；对于 **other** 模板任务，生成器会优先覆写模板参数中的 **memory**，并在能够推导时同步覆写 **maxcore**。

2.2.4.3 gpus <N>

gpus 用于声明当前任务需要的 GPU 数量。该值必须是非负整数，会写入 **.bwrk** 头部并参与 local backend / local queue 的资源解析；未写时按 0 处理。**totgpu** 是别名。

2.2.4.4 gpu_type <text>

gpu_type 用于记录 GPU 型号或站点调度标签，例如 **A100**、**H100** 或站点自定义分区标签。该字段支持运行时任务占位符展开；实际硬件校验由调度系统或站点环境完成。**gputype** 是别名。

2.2.4.5 gpu_mem <MB>

gpu_mem 用于给出 GPU 显存需求提示，单位为 MB。该值必须是非负整数，会写入 **.bwrk** 头部并作为调度或本地队列元数据使用。**gpumem** 是别名。

2.2.4.6 localfirst [<bool>]

```
%control
  localfirst true
```

`localfirst` 是当前任务的 local 优先提示，会作为 `localfirst: true` 写入对应 `.bwrk` 头部。省略布尔值等价于 `true`；也接受 `true` / `false`、`yes` / `no`、`on` / `off` 与 `1` / `0`。

`btrun` 先应用显式 backend、profile、queue、`--no-local` 与资源容量约束；随后，若仍有 local target 能满足核心数、内存和 GPU 请求，且所需资源此刻可以立即取得，就直接选择 local。若 local 资源不足或正忙，`btrun` 不会因为该提示把任务排在 local file queue 后面，而是回到通常的 target / queue 选择策略。因此该开关适合很短的脚本、收尾动作等任务，不会把本应并行提交的计算错误地粘在同一个 local 队列中。

当任务运行在由 `btrun` 生成的调度脚本内时，local 指当前 scheduler allocation 所在的计算节点及其本次分配资源，而不是登录节点上持久化的 local profile。普通任务不会因此自动抢占 local；只有带 `localfirst: true` 的任务才获得上述“可立即运行时优先”行为。

2.2.4.7 `guess true`

当 `guess` 的值设为 `true` 时，如果当前 `%program` 为 Gaussian，系统会默认把 `source` 所对应任务的 `.chk` 检查点文件作为 `%oldchk` 的来源，并在路由关键字中自动补写 `guess=read`；如果当前 `%program` 为 ORCA，则会默认把 `source` 所对应任务的 `.gbw` 文件作为 `%moinp` 的来源，并写入 `Guess MORead` / `AutoStart false`。如果 `guess` 后跟随的是其他非空值，则会被进一步解释为显式的 `guess <task>` 形式。

2.2.4.8 `guess <task>`

```
%control
  guess opt
```

当 `guess` 后面显式给出任务名时，BaneTask 会把该任务视为当前任务的猜测波函数来源。该来源任务会被加入运行时依赖集合，并出现在 `BANETASK_DEPENDENCIES` 环境变量中；对于 Gaussian，生成器会把 `oldchk` 指向该任务目录下对应的 `.chk` 文件，以便复用已有检查点；对于 ORCA，生成器会写入 `%moinp "../<guess-task>/<guess-task>_<origin>.gbw"`，并在 `%scf` 块中写入 `Guess MORead` 与 `AutoStart false`，以复用上游 `.gbw`。

`guess` 的解析规则是：未写 `guess` 行时不启用 guess；值为 `true` 时复用默认来源；其他非空值 (包括 false) 会被视为 `guess <task>`。如需关闭 guess，请删除该行。

同一程序之间的 guess 由对应输入生成器直接写入：Gaussian 写入 `oldchk` 与 `guess=read`，ORCA 写入 `%moinp`、`Guess MORead` 与 `AutoStart false`。跨程序 guess 通过 `guess_converter` 生成预处理命令；`other` 与 `script` 仅保留依赖语义。无论哪种程序，显式 `guess <task>` 都会把该任务纳入运行时依赖。

2.2.4.9 `guess_converter <mokit|multiwfn>`

```
$sp
  %program orca
  %source origin
  %control
    guess opt
    guess_converter mokit
```

`guess_converter` 指定跨程序初猜转换后端。该选项只有在同一 `%control` 中启用 `guess` 时生效；可用值为 `mokit` 与 `multiwfn`，大小写不敏感，`multi-wfn` 与 `mwfn` 会归一化为 `multiwfn`。值为 `none`、`false`、`off`、`no` 或 `0` 时关闭转换器。

当 guess 来源任务与当前任务属于同一程序时，BaneTask 使用该程序的原生 guess 机制，`guess_converter` 不参与输入写入。支持的跨程序转换如下：

来源程序	目标程序	`mokit` 后端	`multiwfn` 后端	目标输入行为
Gaussian	ORCA	`fch2mkl` 后接 `orca_2mkl ... -gbw`	Multiwfn 导出 `.mkl`，后接 `orca_2mkl ... -gbw`	ORCA 输入使用本地 `<inputname>.guess.gbw` 作为 `%moinp`
ORCA	Gaussian	`molden2fch` 后接 `unfchk`	Multiwfn 导出 `.fch`，后接 `unfchk`	Gaussian 输入使用本地 `<inputname>.guess.chk` 作为 `oldchk`
Gaussian	GAMESS	`fch2inp`	Multiwfn 导出 GAMESS `.inp`	目标 GAMESS `.inp` 由转换器生成，并在预处理阶段合并 `%keywords` / `%extrainput` 中的 GAMESS 关键词
ORCA	GAMESS	`molden2fch` 后接 `fch2inp`	Multiwfn 先导出 `.fch`，再导出 GAMESS `.inp`	目标 GAMESS `.inp` 由转换器生成，并在预处理阶段合并 `%keywords` / `%extrainput` 中的 GAMESS 关键词

转换命令写入目标任务的 `pre_comd` 与 `.bwrk` 预处理阶段，而不是在 `banetask` 生成阶段执行。运行环境需能通过 `benv mokit` 或 `benv multiwfn` 加载对应工具；ORCA 目标的 `.gbw` 生成还需要运行期可调用 `orca_2mkl`。ORCA 作为来源时，来源任务目录中需存在 `<guess-task>_<origin>.molden.input`；ORCA workflow 会在主程序完成且 `.gbw` 存在时自动尝试生成该文件。

GAMESS 目标没有独立的外部 guess 文件。对 Gaussian 到 GAMESS 与 ORCA 到 GAMESS 的转换，BaneTask 会把目标任务视为外部输入任务，由转换器在预处理阶段创建 `<inputname>.inp`。随后，`%keywords` 与 `%extrainput` 中可解析为 GAMESS section 的普通关键词会以 `keyword mod` 形式合并进该输入文件；`$DATA` 与 `$VEC` 不作为普通关键词补丁写入。

2.2.4.10 `finalgeom <structure-file>`

```
%control
  finalgeom alllast.xyz
```

`finalgeom` 用于声明当前任务产出的最终几何文件名。该文件需由当前任务的主程序、`%body`、`%process` 或外部脚本写入当前任务目录。文件名可以是绝对路径，也可以是相对路径；相对路径按当前任务目录解析。该文件可以是支持的 XYZ、程序输入文件或程序输出日志；作为下游几何来源时按 `%source` 相同的结构解析规则读取。

声明后，当其他任务通过 `%source <this-task>` 以该任务作为几何来源时：

- 下游从 **<this-task>/<structure-file>** 读取几何信息；
- 下游依赖检查以该结构文件存在性作为完成标志；
- 运行期依赖检查、**%when completed <this-task>**、**completed allothers** 与 finalize 完成判断都会改用该结构文件是否存在作为完成标志；
- 当最终几何文件为 XYZ 时，电荷与自旋按普通 XYZ 解析规则处理；若第二行没有提供可识别的电荷/自旋信息，则使用默认值 **0 1**。

典型用途是让 **%program script** 或外部程序先产出一个标准 XYZ，然后把后续计算任务接到该几何上：

```
$magic_geom_generator
%program script
%control
  finalgeom alllast.xyz
%runner bash
%body << EOF
cmd_that_calculate_xyz_as_you_want export.xyz
cmd_that_write_chg_spin_kv_in_metadata export.xyz
mv export.xyz alllast.xyz
EOF

$opt
%source magic_geom_generator
%keywords "opt freq b3lyp 6-31g*"
```

上例中，**opt** 任务会等待 **magic_geom_generator/alllast.xyz** 出现，并从该文件读取几何；**magic_geom_generator** 本身即使是 **script** 任务、没有常规量化程序日志，只要它可以导出规范 xyz 文件，就可以作为下游 **%source**。

2.2.4.11 waitfreq <bool>

```
%control
  waitfreq false
```

默认情况下，BaneTask 主要检查上游日志尾部的完成标志来判断任务是否结束。当该项设置为 **false** 时，系统会改为扫描整个日志文件以确认是否出现正常结束标志。这一选项适合下游仍需继续优化或继续处理的场景：即使上游频率任务在末尾报告了虚频，只要日志中已经出现过正常结束信息，当前任务仍然可以继续生成。

2.2.4.12 alert <bool>

```
%control
  alert false
```

alert 控制 workflow 对主命令失败的处理方式，默认值为 **true**：

- **alert true**：workflow 会保留主命令返回码作为最终退出码；若已配置 **BANETASK_ALERT_NTFY**，主命令失败时还会向对应 ntfy 主题发送通知。
- **alert false**：workflow 仍会写出 **BANETASK_MAIN_RC** 和 **BANETASK_MAIN_OK**，跳过 ntfy 通知，且最终退出码固定为 **0**。

该设置适合那些“允许失败但仍希望继续归档或后处理”的辅助任务。若需要让外层调度

系统严格感知失败，应保持默认的 **alert true**。

2.2.4.13 verbosity p|t

verbosity 用于控制 Gaussian 路由行的前缀形式，其中 **p** 对应 **#p**，**t** 对应 **#t**；若未设置或为空，则默认使用普通的 **#**。这一选项主要用于在不同详细程度的输出之间做快速切换。

2.2.4.14 geom check

```
%control
  geom check
```

geom check 只对 Gaussian 生效，用于从检查点文件中直接读取几何。启用后：

- 当前任务会复用依赖任务或显式 **guess <task>** 所对应的检查点文件；
- 若路由行中尚未给出 **geom=check** / **geom=allcheck**，生成器会自动补写 **geom=check**；
- 若找不到可用的依赖/guess 检查点来源，输入生成会直接报错。

该选项适合在 Gaussian 路径上执行纯检查点续算、直接读取上游优化后几何的任务。

2.2.4.15 externalinp <bool>

```
%control
  externalinp true
```

当 **externalinp** 设为 **true** 时，BaneTask 会把当前任务视为“输入文件由外部流程准备、主程序仍由 workflow 执行”的任务：

- 不生成托管输入文件；
- 仍然照常写出 workflow 主运行命令；
- 若当前目录下尚未出现匹配的输入文件，workflow 会直接采用约定文件名 **任务名 _原始名.后缀** 来生成运行脚本；
- 因而可以把“生成该正确文件名输入文件”的责任交给 **%preprocess**、外部脚本或其他外部程序。

这适用于以下场景：任务输入由外部脚本准备，但 workflow、调度命令和后处理仍交给 BaneTask 管理。典型做法是让 **%preprocess** 先产出例如 **opt_origin.gjf**、**sp_origin.inp** 或其他符合环境配置后缀的文件，再由 workflow 中的主命令实际执行。

2.2.5 %interface

%interface 用于把一个任务声明为 Gaussian External 接口提供者，或让另一个 Gaussian 任务消费该接口。接口提供者只描述外部量子化学程序的参数；它本身不生成独立任务目录、输入文件或 workflow。接口消费者在生成 Gaussian 输入时读取 provider 配置，并在当前任务目录写出 **qc2g16_external** 配置文件和 External 启动脚本。

语法如下：

```
%interface provider
%interface consume <provider-task>
```

支持以下组合：

- provider：gaussian、xtb、orca、gamess、amesp

- consumer：gaussian

其他设定：

- 生成 consumer 时，BaneTask 会在 `BANETASK_ENVCONF_PATH` 中查找 provider 对应的现有环境配置。Gaussian/g16 使用 `g16.conf`，XTB 使用 `xtb.conf`，ORCA 使用 `orca.conf`，GAMESS 按顺序查找 `gamess.conf`、`gms.conf`，AMESP 使用 `amesp.conf`；不会创建或查找额外的 `g16_backend.conf`。
- provider 环境配置必须同时提供 `ENV_SETUP` 和非空 `RUN_CMD_TEMPLATE`。配置文件不存在、缺少任一字段、`RUN_CMD_TEMPLATE` 为空或模板没有引用输入文件时，consumer 在输入生成阶段立即失败。
- External 启动脚本按运行平台生成：Linux 和其他 POSIX 环境使用 Bash 脚本，Windows 使用 CMD 批处理脚本。启动脚本先执行 provider 的 `ENV_SETUP`，随后启动 `qc2g16_external`。外部量化程序的主调用命令来自同一配置文件的 `RUN_CMD_TEMPLATE`，因此程序路径、wrapper、MPI 启动器和重定向策略仍沿用普通任务的 env 配置语义；Windows 配置应使用 CMD 语法。
- 资源按用途分别处理：Gaussian 输入使用 consumer 的 `totcore` / `totmem`；`qc2g16_external` 配置使用 provider 的 `totcore` / `totmem`，未设置时回退到 consumer；生成 workflow 时，核数与总内存分别取 consumer 和 provider 的较大值，以保证调度器申请的资源能够覆盖两边程序。某一侧未设置的值按 `0` 参与比较；两侧都未设置时继续使用环境配置默认值。
- consumer 的 `%keywords` 可使用 `[external]`，生成时替换为当前任务目录中的 External 启动脚本路径。
- consumer 的 `%keywords` 也可使用 `[external_config]` 和 `[external_launcher]`，分别引用生成的 `qc2g16_external` 配置文件名和启动脚本文件名。
- provider 的 `%args`、`%extrainput` 与 `%extrainputs` 中可使用 `[charge]`、`[spin]`、`[uhf]`、`[xyz_file]`、`[derivative_order]`、`[derivative_order_int]`、`[derivative_keyword]`；生成 `qc2g16_external` 配置时会映射为 External 运行期占位符。
- `RUN_CMD_TEMPLATE` 至少必须引用 `${ACTUAL_INPUT_FILE}`、`${ACTUAL_INPUT_FILENAME}` 或 `${ACTUAL_INPUT_STEM}` 之一。`${ACTUAL_OUTPUT_FILE}`、`${ACTUAL_INPUT_FILENAME}`、`${ACTUAL_OUTPUT_FILENAME}`、`${ACTUAL_INPUT_STEM}`、`${ACTUAL_OUTPUT_STEM}`、`${FILE_NAME}`、`${BASENAME}`、`${WORKDIR}`、`${CORES}`、`${TYPE}` 会映射到 `qc2g16_external` 的运行期占位符。
- 只有 XTB provider 需要把运行期参数放到启动命令中。BaneTask 将 `%args`、自动补齐的 charge/UHF 参数以及导数参数统一合并到现有通用 `${ARGS}` / `${EXTRA_ARGS}` 占位符；XTB 的 `RUN_CMD_TEMPLATE` 若没有这两个占位符之一，输入生成直接失败。
- ORCA、GAMESS、AMESP 和 Gaussian 的导数类型写入各自输入文件，不要求命令模板声明参数占位符。不会引入 `${DERIVATIVE_KEYWORD}` 之类的后端专用 env 占位符。
- 导数任务只使用统一占位符 `{{derivative_keyword}}`；DSL 中可写 `[derivative_keyword]`。具体展开值由 `qc2g16_external` 根据 `[program] kind` 和 Gaussian 请求的导数阶数决定。xTB 的能量、梯度和 Hessian 请求分别展开为空字符串、`--grad` 和 `--hess --grad`；能量从 `energy` sidecar 读取。
- Gaussian provider 的 `%keywords` 必须给出后端 Gaussian 的方法、基组及其他 route 设置；若未显式包含 `[derivative_keyword]` 或 `{{derivative_keyword}}`，生成器会自动追加。主计算调用直接复用 `g16.conf` 的 `RUN_CMD_TEMPLATE`；计算成功后由 `%interface` 固定调用 `formchk` 生成 `{{input.stem}}.fchk`，不把 `formchk` 塞入 env 的

命令模板。

- ORCA provider 的 `%keywords` 作为 ORCA 主关键词使用。若其中未显式包含 `[derivative_keyword]` 或 `{{derivative_keyword}}`，生成器会自动追加 `{{derivative_keyword}}`，使 Gaussian 能按能量、梯度或 Hessian 请求分别生成 `SP`、`EnGrad` 或 `EnGrad Freq` 输入。
- GAMESS provider 的 `%keywords` 与 `%extrainput` 作为 GAMESS section 文本使用。BaneTask 在生成 `qc2g16_external` 配置时会增加时间戳后缀，避免并发任务在 GAMESS `USERSCR` 中发生同名冲突。`qc2g16_external` 生成输入时会按 Gaussian 请求自动设置 `$CONTRL RUNTYP=ENERGY`、`GRADIENT` 或 `HESSIAN`。
- AMESP provider 的 `%keywords` 作为 `!` 方法行，`%extrainput` 作为 `%` 指令或原生 section。导数占位符按请求展开为空、`force` 或 `numfreq`；Hessian 使用 `numfreq`，以便一次返回 Gaussian External 所需的能量、平衡结构梯度和 Cartesian Hessian。
- XTB provider 会自动补齐 `--chrg {{charge}}` 与 `--uhf {{uhf}}`，除非 `%args` 已显式给出对应选项。
- XTB 的导数参数同样并入 `${ARGS}` / `${EXTRA_ARGS}`：Gaussian 请求梯度时加入 `--grad`，请求 Hessian 时加入 `--hess --grad`。

示例：

```
$interface-gfn2-xtb
  %program xtb
  %args --gfn 2 --chrg [charge] --uhf [uhf]
  %interface provider

$external-xtb
  %source origin
  %interface consume interface-gfn2-xtb
  %keywords "opt(nomicro,modr) external='[external]' freq"
  %extrainputs << EOF
1 2 F
EOF
```

Gaussian provider 示例（外层 Gaussian 调用另一套 g16）：

```
$interface-g16-backend
  %program g16
  %keywords "wB97XD/def2TZVP nosymm"
  %control
    totcore 16
    totmem 32000
  %interface provider

$external-g16
  %source origin
  %program g16
  %interface consume interface-g16-backend
  %keywords "opt(nomicro,modr) external='[external]' freq"
```

Gaussian provider 直接复用 `BANETASK_ENVCONF_PATH` 中现有的 `g16.conf`：`ENV_SETUP` 初始化 Gaussian 环境，`RUN_CMD_TEMPLATE` 只负责执行主 Gaussian 命令。`%interface` 在主

命令成功后固定调用同一环境中的 **formchk**；不需要也不允许通过额外的 **g16_backend.conf** 改变这套语义。

ORCA provider 示例：

```
$interface-orca
  %program orca
  %keywords "r2SCAN-3c def2-SVP"
  %interface provider

$external-orca
  %source origin
  %interface consume interface-orca
  %keywords "opt(nomicro,modr) external='[external]' freq"
```

GAMESS provider 示例：

```
$interface-gamess
  %program gamess
  %keywords "$CONTRL SCFTYP=RHF $END"
  %extrainput << EOF
 $BASIS GBASIS=STO NGAUSS=3 $END
EOF
  %interface provider

$external-gamess
  %source origin
  %interface consume interface-gamess
  %keywords "opt(nomicro,modr) external='[external]' freq"
```

AMESP provider 示例：

```
$interface-amesp
  %program amesp
  %keywords "B3LYP def2-SVP [derivative_keyword]"
  %interface provider

$external-amesp
  %source origin
  %interface consume interface-amesp
  %keywords "opt(nomicro,modr) external='[external]' freq"
```

生成 consumer 任务时，任务目录中会额外出现两类 sidecar：

- **<origin>_<consumer>.qc2g16_external.conf**：**qc2g16_external** 配置文件。Gaussian provider 读取 **{{input.stem}}.fchk**，XTB provider 读取 **gradient** / **hessian**，ORCA provider 读取 **{{input.stem}}.engrad** / **{{input.stem}}.hess**，GAMESS provider 读取 **{{input.stem}}.dat** 或 punch/dat sidecar，AMESP provider 直接解析主输出 ***.aop**。
- **<origin>_<consumer>.external.sh** 或 **<origin>_<consumer>.external.cmd**：传给 Gaussian **External(...)** 的启动脚本。POSIX 环境生成 **.external.sh**，Windows 生成 **.external.cmd**。该脚本先执行 provider 环境配置中的 **ENV_SETUP**，再调用 **qc2g16_**

`external`；外部程序命令由同一配置中的 `RUN_CMD_TEMPLATE` 写入前一个 sidecar。

外部量子化学程序的运行目录固定为 consumer 任务目录下的 `./external`。`qc2g16_external` 会自动创建该目录；生成的外部程序输入、标准输出、梯度、Hessian、dat/punch 及程序附带文件均保存在其中，避免污染 consumer 任务的常规目录。External 配置文件、启动脚本、Gaussian 输入与 Gaussian 输出仍位于 consumer 任务目录。

接口 provider 不参与 `completed allothers` 的阻塞集合，也不会作为独立运行任务进入正常 workflow。若 provider 任务名被直接写入 `%when completed <provider>`，仍会被视为普通显式依赖名；此类写法通常不应使用。

2.2.6 %when

`%when` 用于控制任务是否进入生成阶段，是 BaneTask 中最重要的条件判定机制之一。条件满足时任务进入生成阶段；条件不满足时，输入文件、命令脚本和 workflow 保持未生成状态。它在流程控制上的作用相当于“带数据依赖的门闩”。

2.2.6.1 书写形式

`%when` 既支持多行条件块，也支持单行写法。多行形式更适合表达一组需要同时满足的约束，而单行形式则适合简单判断。

```
%when
  completed opt
  {{ opt.nimag }} = 1
  {{ opt.imag1 }} < -500
```

也支持单行形式：

```
%when {{ energy }} < -100
```

2.2.6.2 语法规则

`%when` 语法如下：

```
when-block    := when-line+
when-line     := or-expr
or-expr       := and-expr ("or" and-expr)*
and-expr      := unary-expr ("and" unary-expr)*
unary-expr    := leaf | "(" or-expr ")"
leaf          := "defined" value
               | "completed" task_name_or_pattern
               | operand op operand
operand       := literal | placeholder
placeholder   := "{{" ref "}}"
ref           := task_name "." key | key
op            := "=" | "==" | "!=" | "<" | "<=" | ">" | ">="
```

求值时遵循以下规则：多个 `%when` 行之间按 **AND** 处理；单行中 `and` 的优先级高于 `or`；可以使用括号显式分组；`<`、`<=`、`>`、`>=` 要求比较两侧都能被解析为数值；而 `=`、`==`、`!=` 在两侧都为数值时执行数值比较，否则执行字符串比较。

2.2.6.3 defined [NAME]

```
defined [ENABLE_TS]
```

当 [NAME] 在变量替换之后已经解析为具体值时，该条件即判定为真。因此，defined 最常用于检测某个可选 define 是否已经提供，从而决定是否启用相应的任务分支。

2.2.6.4 completed <task>

```
completed opt
```

completed <task> 用于检查指定上游任务是否已经完成。默认情况下，它要求目标任务的日志文件存在且已正常结束；完成判断读取任务目录中的运行产物，适合用于生成阶段的即时条件控制。

若目标任务声明了 %control finalgeom <xyz-file>，completed <task> 会改为检查该任务目录中的 <xyz-file> 是否存在。这一规则同样适用于由通配符匹配到的任务。

completed 的目标也可以写成 shell 风格的任务名通配符，支持 *、? 与 [...]：

```
%when completed opt_*
```

当目标含有通配符时，系统会在当前任务文件的完整任务集合中匹配任务名。若没有匹配项，条件判定为不通过；若匹配项中有任务因为 %when 被当前计划裁掉，这些非活动分支会从当前条件的阻塞集合中排除；其余匹配任务必须满足各自的完成判据：声明 finalgeom 的任务检查指定 XYZ 文件，其余任务检查正常结束日志。

2.2.6.5 completed allothers

```
completed allothers
```

completed allothers 是 completed 的特殊目标，用于等待“当前任务文件中除本任务外的其他有效任务”全部完成。这里的“有效任务”指仍处于活动状态、且没有被 %when 直接裁掉的分支；已经被条件排除的分支会从该条件的阻塞集合中排除。

它适合用于尾部汇总、统一导出或显式收尾任务，例如：

```
$final_report
  %program script
  %runner bash
  %when completed allothers
  %body echo done
```

补充规则如下：

- 该条件依赖当前任务文件的完整任务上下文，因此只在正常解析整份 .bt / .projbt 时有意义。
- 在 matrix 重写过程中，它会保持为特殊目标本身。
- 若多个任务同时互相等待 completed allothers，会形成尾部僵持；btool lint 会对这类模式给出 deadlock 警告。

2.2.6.6 快照占位符 {{ ... }}

快照占位符用于直接读取上游任务的 .bane.result.kv 内容，因此可以把已提取的能量、虚频数、激发态信息等结果引入生成条件中。其基本形式如下：

```
{{ opt.energy }}
{{ opt.nimag }}
{{ opt.tddft.total_energy }}
```

当当前任务只有一个默认依赖时，可以省略任务名前缀：

```
{{ energy }}
{{ nimag }}
{{ imag1 }}
{{ converged }}
```

.bane.result.kv 会写入统一规范属性目录中的 canonical qc.* 标量字段。由于 qc.energy.primary、qc.vibration.imaginary_count 这类字段本身包含点号，若当前任务只有一个默认依赖，也可以直接省略任务名前缀：

```
%source opt
%when
  {{ qc.vibration.imaginary_count }} = 0
  {{ qc.energy.primary }} < -100
```

当存在多个上游依赖时，必须显式写成 {{ task.field }}，否则条件求值会报错。此时 BaneQC canonical 字段应写成：

```
%when {{ opt.qc.vibration.imaginary_count }} = 0
```

2.2.6.7 当前可直接使用的常用字段

.bane.result.kv 中的任意字段都可以直接引用。为了便于在条件中书写，系统还内置了若干常用别名，其查找顺序如下：

- energy：依次查找 energy、tddft.total_energy、thermochemistry.gibbs_free_energy、qc.energy.primary、scf_energy、qc.energy.scf。
- free_energy：依次查找 free_energy、thermochemistry.gibbs_free_energy、qc.thermochemistry.gibbs_energy。
- nimag：依次查找 nimag、frequency.imaginary_count、qc.vibration.imaginary_count。
- imag1、imag2、…：依次映射到 imagN 或 frequency.imaginary_frequencies.N。

示例：

```
%when
  completed freq
  {{ freq.nimag }} = 1
  {{ freq.imag1 }} < -500
```

2.2.6.8 快照自动补建

当 %when 需要读取上游快照,而对应的 .bane.result.kv 缺失或时间戳早于日志文件时,BaneTask 会自动尝试调用 btproc result 重新生成快照。因此，%when 不仅能用于首次生成，也

适用于已有目录的补跑、恢复和增量更新场景。快照补建只通过 `btproc result` 完成；如果 `btproc result` 不可用或提取失败，依赖加载会直接报错。

2.2.6.9 派生值占位符 `{{ derived(...) }}`

任务块可以读取位于其前方的项目级 `@derive` 输出。推荐使用两个参数分别指定 derive id 和输出名：

```
{{ derived("limits", "threshold") }}
```

也可使用单参数选择器或函数别名：

```
{{ derived("limits.threshold") }}
{{ derive_value("limits", "threshold") }}
```

`@derive` 必须出现在引用它的任务块之前。普通完整模式会在准备第一个引用任务前保证该 derive 的结果可用；需要计算时，derive 及其 derive 依赖按依赖顺序执行。结果文件位于 `Results/Derived/<derive-id>.derived.json`。

占位符可用于 `%when`，也可用于 `%keywords`、`%args`、`%extrainput`、`%body`、`%runner`、模板参数、前后处理命令和 `%mod` 值等任务生成阶段解析 `{{ ... }}` 的文本字段。任务名、`@derive` 声明和 `%source` 依赖声明不使用该占位符。

任务占位符只接受 `scalar` 和 `text_scalar` 输出。数值标量展开为结果中保存的数值，不附加单位；文本标量按原文展开。序列和表不能直接写入任务字段，应先在 Derive DSL 中归约或选择为标量。

```
@derive limits
  emit threshold = 12.5
@end

$consumer
  %source origin
  %program script
  %runner bash
  %when {{ derived("limits", "threshold") }} > 10
  %body << SH
printf 'threshold=%s\n' '{{ derived("limits", "threshold") }}'
SH
```

`--comd-only` 或 `--no-derives` 不执行 derive；这两种模式下，引用所需的 `.derived.json` 必须已经存在。普通完整模式中，任务所需的 derive 若因缺少输入结果而无法生成，引用任务会失败，因为其文本字段无法完成替换。

2.2.7 `%keywords`

`%keywords` 的内容由 `%program` 决定。本章只定义它是任务级程序输入字段；各后端的语法、合并顺序、默认值和限制见下表。

程序	说明位置
Gaussian	Gaussian

程序	说明位置
ORCA	ORCA
GAMESS	GAMESS
AMESP	AMESP
MOPAC	MOPAC
BDF	BDF
FCclasses	FCclasses
MRCC	MRCC
MAPLE	MAPLE
XTB	XTB；规范字段为 **%args**
script	script；规范字段为 **%runner**
other	other；规范字段为 **%template** 与 **%param**

2.2.8 %extrainput

%extrainput 的写入位置和解析方式同样由 **%program** 决定；**%extrainputs** 是等价输入别名。具体格式见下表。

程序	说明位置
Gaussian	Gaussian
ORCA	ORCA
GAMESS	GAMESS
AMESP	AMESP
MOPAC	MOPAC
BDF	BDF
FCclasses	FCclasses
MRCC	MRCC
MAPLE	MAPLE
XTB	XTB

2.2.9 %args

%args 是 **%program xtb** 的正式命令行参数入口，用于声明传给 XTB 主程序的参数。该字段由 XTB 输入生成器消费；如果非 XTB 任务声明了 **%args**，解析器会给出提示并忽略该字段。

```
$xtb_opt
  %program xtb
  %source origin
  %args --gfn 2 --opt tight
```

规则如下：

- `%args` 可以写成单行，也可以写成多行块；多行块会按空格拼接成一组命令行参数。
- 若任务没有显式 `%program`，但只提供 `%args`、且没有其他程序类型的正式输入描述，系统会自动按 `%program xtb` 处理。
- `%program xtb` 可把 `%keywords` 作为 `%args` 读取；任务文件应直接使用 `%args`。
- `%args` 与 XTB 的 xcontrol sidecar 正文中可使用输入生成阶段占位符：`[xyz_file]`、`[charge]`、`[spin]` 与 `[uhf]`。其中 `[uhf]` 按应用 `%mod` 后的自旋多重度计算为 `spin - 1`，适合写作 `--uhf [uhf]`。
- `%extrainput` 会写入 XTB 的 xcontrol sidecar；`%mod freeze` 也可能生成约束 sidecar。
- 当需要 xcontrol sidecar 时，BaneTask 会把对应 `--input <task>_<origin>.xcontrol` 交给 workflow；因此 `%args` 中不应再手写 `--input` 或 `--input=...`。

2.2.10 %template

`%template` 只在 `%program other` 下生效，用于指定要加载的模板基名（不含后缀）：

```
%template xtb_opt
%template "orca_single_point"
```

规则如下：

- 生成 `other` 输入时必须提供模板名；若缺失，生成器会报错，`btool lint` 也会给出 `missing-template`。
- `%template <name>` 与 `job=<name>` 语义等价；规范写法是 `%template`。
- 模板后缀由 `%param suffix=<ext>` 或替代参数决定；若未给出，则默认使用 `inp`。

2.2.11 %param

`%param` 只在 `%program other` 下生效，用于提供简单的 `key=value` 参数表。它既可以写成单行，也可以写成多行块：

```
%param prog=xtb suffix=in charge=0
```

```
%param
  prog=orca
  functional="wB97M-V"
  basis="def2-TZVP"
```

规则如下：

- 参数按简单 `key=value` 解析；值可以使用单引号或双引号包围，以保留空格。
- 若同名参数出现多次，后出现的值覆盖前值。
- `%param` 中的值会覆盖 `%keywords` 替代参数中的同名键，然后由模板默认段补齐缺失项；最后 `%control totcore/totmem` 覆写资源相关参数。
- 常见约定键包括：`prog`（决定 workflow 所用环境配置）、`suffix`（输出后缀）以及模板自身定义的任意业务参数。

2.2.12 %runner

%runner 只在 %program script 下生效，是 script runner / shebang 的规范指令：

```
%runner /usr/bin/env bash
%runner "/bin/bash"
```

规则如下：

- %runner 优先级高于 %keywords 中的替代 runner。
- 若 %runner 与 %keywords 中的替代 runner 都为空，生成器默认使用 /bin/bash。
- 在 Bash 启动器下，bash / sh runner 会生成单文件脚本；python、python3、powershell、pwsh 或未知 runner 会生成 launcher + payload。
- 在 Cmd 启动器下，cmd runner 会生成单文件 .bat；bash、sh、python、powershell、pwsh 会生成 .bat launcher 并调用对应 payload。

2.2.13 %body

%body 只在 %program script 下生效，是脚本正文指令。它支持单行、heredoc 和文本片段三种形式：

单行内联：

```
%body echo hello [inputname]
```

heredoc：

```
%body << EOF
echo begin
echo "[inputname]"
EOF
```

文本片段：

```
%body
  &INCLUDE "scripts/run.sh"
```

规则如下：

- heredoc 形式以自定义终止符结束；若缺失终止符，解析器与 btool lint 都会给出提示。
- 文本片段形式按 text 上下文读取，保留 shebang、注释、shell 变量和字面量 &INCLUDE；heredoc 内的 &INCLUDE 不展开。
- 正文中仍可使用脚本生成器支持的占位符,例如 [xyz_file]、[geom]、[charge]、[spin]、[uhf],以及通用的 [rootname]、[taskname]、[inputname]、[originname]、[source-name] 与 [define] 变量。
- 若 %body 为空且未启用 externalinp，脚本任务仍会生成带 header 的脚本文件，但 btool lint 会给出 missing-script-body 警告。
- 当脚本任务准备作为其他任务的几何来源时，可在 %control 中声明 finalgeom <xyz-file>；脚本正文需要在任务目录中写出该 XYZ，后续 %source <script-task> 会以它为唯一几何来源和完成标志。

2.2.14 %batch

%batch <N> 用于多帧 XYZ 批处理，并在当前父任务目录内部生成 **.batch/** 工作区。每个 frame 是父任务内部的一个可独立运行、独立记录状态、独立重跑的子任务。

已接入的 program：**script**、**xtb**、**gaussian**、**orca**、**gamess**、**amesp**、**mopac**、**bdf**、**mrcc**、**maple**、**other**。

script 示例：

```
$frame_analyze
  %program script
  %source xyz=traj
  %runner /usr/bin/env bash
  %batch 4
  %body << EOF
echo "frame=${BT_FRAME} xyz=${BT_XYZ} charge=${BT_CHARGE} uhf=${BT_UHF}"
my_analyzer "${BT_XYZ}" > "analysis.out"
EOF
```

XTB 示例：

```
$xtb_scan
  %program xtb
  %source xyz=traj
  %batch 4
  %args --gfn 2 --opt normal
```

需要添加详细规则时，可把 **%batch** 写成块形式：

```
$g16_scan
  %program gaussian
  %source xyz=traj
  %batch
    lane 4
    collect "*.fchk"
    collect required "*.cube" as cubes/[frame_name]/
    collect "*.wfn" as wavefunctions/[frame_name]_[basename]
```

collect 语法为 **collect [required|--required] < 源模式 > [as|to < 目标模式 >]**。源模式相对每个 **.batch/frame_XXXXXX/** 目录解释；目标模式相对父任务 **final/** 解释。未写目标模式时默认收集到 **final/collect/[frame_name]/[basename]**。

源模式和目标模式支持 frame 占位符：**[frame]**（1-based frame 号）、**[index]**（0-based index）、**[frame_name]**、**[input]**、**[input_stem]**、**[output]**、**[output_stem]**、**[program]**。**[output]** 是程序配置中 **OUTPUT_SUFFIX** 对应的规范结果文件，例如输入 **calc.gjf** 配合 **OUTPUT_SUFFIX=log** 时为 **calc.log**；**[output_stem]** 在该例中为 **calc**。目标模式还支持匹配文件占位符：**[basename]**、**[stem]**、**[ext]**/**[extension]**。目标以 **/** 结尾时会自动追加匹配文件名。为了避免越界复制，绝对路径和含 **..** 的源/目标路径会被拒绝。

2.2.14.1 构象聚类

当成功 frame 能汇总为 **final/optimized_traj.xyz** 时，可在 **%batch** 块中增加 **cluster** 规则。banetask 提供类似 molclus 的 isostat 距离矩阵聚类和 RMSD 聚类两种选择。其中距离矩阵聚类是按照 sobereva 老师对聚类算法的描述进行的实现。距离矩阵聚类示例：

```
$xtb_scan
  %program xtb
  %source xyz=traj
  %batch
    lane 8
    cluster dist geo=0.10 energy=0.25 temperature=298.15 accum=0.95
  %args --gfn 2 --opt normal
```

不依赖能量的 RMSD 聚类示例：

```
%batch
  lane 8
  cluster rmsd geo=0.30 energy=off
```

规范语法为：

```
cluster dist|rmsd
        [geo=<angstrom>]
        [energy=<kcal/mol>|off]
        [temperature=<kelvin>]
        [min=<weight>]
        [accum=<weight>]
```

参数含义如下：

- **dist**：按无序元素对分组全部原子间距离，在每组内排序，以对应分量的最大绝对差作为几何距离。默认 **geo=0.10** Å、**energy=0.25** kcal/mol，并要求每个输入 frame 都有有限能量。
- **rmsd**：按固定原子对应关系执行 Horn 刚体叠合，再计算 RMSD。默认 **geo=0.10** Å；双方都有能量时同时应用默认 **energy=0.25** kcal/mol，能量缺失时仍可按几何聚类。
- **energy=off**：完全忽略能量，只允许与 **rmsd** 组合；此时不能再使用 **min** 或 **accum**。
- **temperature**：Boltzmann 权重温度，默认 **298.15** K，必须为有限正数。
- **min**：丢弃权重低于该值的簇，范围为 **[0, 1)**。
- **accum**：按代表能量从低到高保留簇，直到累计权重首次达到该值，范围为 **(0, 1]**。

聚类算法标识为 **greedy-representative-v1**。frame 按优化轨迹顺序处理，并加入第一个同时满足所有启用阈值的已有簇；阈值比较采用严格小于。能量可用时，簇内最低能量 frame 作为代表结构；没有能量时保留该簇最先接收的 frame。代表结构更新后，后续 frame 会与新的代表结构比较，因此结果可能依赖输入顺序。

dist 不区分镜像异构体，且可能把距离同谱结构合并；**rmsd** 不搜索等价原子置换或分子对称性。两种判据都要求所有 frame 的原子数和逐位置元素顺序一致。

所有 frame 都有能量时，簇按代表能量从低到高排序，并计算相对能量与 Boltzmann 权重。权重只使用代表结构能量，不把 **cluster_size** 作为简并度乘入配分函数。使用 **min** 或 **accum** 时，任一输入 frame 缺少能量都会使聚类失败。

batch driver 与 frame launcher 根据 workflow shell 选择文件格式：Bash 使用 **run_batch.sh** 和 **run.sh**；Windows Cmd 使用 **run_batch.bat**、按 lane 生成的 **lane_XXX.bat**，以及每帧 **run.bat**。Cmd lane worker 会为每个 frame 启动独立的 inner **cmd.exe**，因此 frame 脚本中的 **exit** 不会终止 lane worker；worker 仍会记录返回码、写入 **status.json**，并生成 lane 完成哨兵。

batch 的目录结构如下（括号内为 Windows Cmd 文件名）：

```
xtb_scan/
  .batch/
    manifest.json
    run_batch.sh                  # Windows Cmd: run_batch.bat
    lane_000.bat                  # 仅 Windows Cmd；每个活跃 lane 一个 worker
    frame_000001/
      input.xyz
      run.sh                      # Windows Cmd: run.bat
      <input_stem>.<OUTPUT_SUFFIX>  # 程序规范结果文件
      run.log                       # 启动层标准输出和标准错误
      status.json
      input.xcontrol              # xtb 可选
      xtbopt.xyz                  # xtb 运行产物
    frame_000002/
      input.xyz
      run.sh                      # Windows Cmd: run.bat
      run.log
      status.json
  final/
    batch.summary.json
    failed_frames.txt
    optimized_traj.xyz            # 支持的优化程序可提取结构时生成
    optimized_traj.frames.txt
    clustered_traj.xyz            # 请求聚类且成功时生成
    cluster.summary.json          # 请求聚类且成功时生成
```

规则如下：

- batch 在生成逐帧输入前按 lane 数切分任务级资源：**frame_totcore=max(1, totcore/lanes)**，**frame_totmem=max(1, totmem/lanes)**，均采用整数除法。Gaussian 的 **%nproc/%mem**、ORCA 的 **%pal/%maxcore**、AMESP 的 **%npara/%maxcore**、MOPAC 的 **THREADS** 和 other 模板中的 **nprocs/memory/maxcore** 使用切分后的值，并与 **BT_CORES_PER_LANE** 保持一致。用户在原始 **%extrainput** 中手写的程序资源指令不会被强制改写，应自行避免与 batch lane 配置冲突。
- 每个 frame 的规范程序输出名按实际输入文件名与 program conf 中的 **OUTPUT_SUFFIX** 推导，形式为 **<input_stem>.<OUTPUT_SUFFIX>**。**SUFFIX** 模式会移除完整配置后缀，**INPUT_FILENAME** 模式按固定 basename 的普通 stem 规则处理；无后缀输入（例如 **MINP**）保留完整文件名作为 stem。可重定向输出的程序应在 **RUN_CMD_TEMPLATE** 中使用 **${ACTUAL_OUTPUT_FILE}**；会自行生成同 stem 结果文件的程序可以不引用该变量。**run.log** 只保存 frame 启动层的标准输出和标准错误，几何与能量提取读取规范程序输出。
- 必须显式使用 **%source xyz=<name>**，并且不要在该来源上写 **frame=**；批处理会消费全部帧。

- 每个 frame 都在 `.batch/frame_XXXXXX/` 下运行，并拥有输入文件、规范程序输出、平台对应的 `run.sh` 或 `run.bat`、`run.log` 与 `status.json`。
- 父任务 workflow 在 Bash 下调用 `.batch/run_batch.sh`，在 Windows Cmd 下调用 `.batch/run_batch.bat`。每个后端会为当前 frame 写出 launcher；没有专用 launcher 时，按对应程序环境配置生成。
- `script` batch 的 `%body` 会被逐帧渲染成 `frame_XXXXXX/run.sh` 或 `run.bat`，并在该 frame 目录中执行；因此 `BT_XYZ=input.xyz`。Cmd launcher 与普通 script 任务使用相同的 launcher + payload 规则。
- script batch 会提供环境变量：`BT_TASK_DIR`、`BT_BATCH_DIR`、`BT_FRAME_DIR`、`BT_BATCH`、`BT_BATCH_FRAME_COUNT`、`BT_LANE`、`BT_INDEX`、`BT_FRAME`、`BT_FRAME_NAME`、`BT_XYZ`、`BT_INPUT_XYZ`、`BT_OUTPUT_FILE`、`BT_CHARGE`、`BT_SPIN`、`BT_UHF`、`BT_CORES_PER_LANE`。
- batch 会为每帧解析 charge/spin；当前帧缺失时回退 origin，origin 也缺失时使用 `0 1`。`[uhf]` 与 `BT_UHF` 始终按 `spin - 1` 计算。
- `[charge]`、`[spin]`、`[uhf]` 在 batch 中按 frame 延迟渲染，因此不同 frame 可以拥有不同电荷和自旋。
- `xtb` batch 会沿用 `xtbopt.xyz` 汇总 `final/optimized_traj.xyz`；`gaussian`、`orca`、`gamess/gms`、`amesp`、`bdf`、`mrcc`、`maple`、`mopac` batch 会对成功 frame 调用 `btproc fork collect` 转换程序输出并追加到同一个轨迹文件，XYZ 注释行包含可获得的 `charge spin E=<Hartree>`。
- `final/optimized_traj.frames.txt` 与优化轨迹逐帧对应，保存原始 `frame_XXXXXX` 名称。失败 frame 或没有可提取几何的成功 frame 不会进入这两个文件。
- 请求 `cluster` 时，batch 在汇总优化轨迹后生成 `final/clustered_traj.xyz` 与 `final/cluster.summary.json`。代表轨迹的注释行记录原 batch frame、输出簇序号、原始簇序号、簇大小以及可用的能量和权重。
- `cluster.summary.json` 记录算法、判据、阈值、能量策略、输入 frame 数、全部簇、保留簇、代表 frame 和成员 frame；`batch.summary.json` 同时记录 `cluster_requested` 与 `cluster_status`。
- 没有可聚类轨迹、输入结构不兼容、所需能量缺失或聚类产物写出失败时，batch 主阶段返回失败；已生成的 `batch.summary.json` 和 frame 状态仍保留，父任务 `%process` 仍会执行。
- 聚类不会覆盖 `optimized_traj.xyz`，也不会自动改变任务最终几何。下游需要使用代表结构轨迹时，应显式写 `%control finalgeom final/clustered_traj.xyz`。
- XTB batch 的 `%args` 会逐帧渲染；如果用户没有显式写 `--chrg` 或 `--uhf`，生成器会自动追加当前帧的 `--chrg <charge>` 与 `--uhf <uhf>`。
- batch workflow 会默认跳过 `status.json` 已标记为 success 的 frame，并在 `final/` 下生成 `batch.summary.json`、`failed_frames.txt`，以及可提取几何时的 `optimized_traj.xyz` 和 frame 映射。

2.2.15 %mod

`%mod` 用于在生成当前任务输入时临时调整电荷、自旋多重度和几何，也可配置 Gaussian、ORCA、XTB 等后端已有的基组、冻结、分层或片段输入。修改只写入当前任务准备结果，不回写原始输入文件、上游任务结果或 case 根目录中的源结构。

单几何任务继续使用原有写法：

```
%mod
  charge 1
  spin 2
  geom select 1-20
```

使用命名 **%source** 时，**charge**、**spin** 和 **geom** 可以加与几何 binding 相同的 slot：

```
$multi
  %source
    geom Raw
    geom.end Product
  %mod
    charge 1
    geom select 1-30
    charge.end 0
    spin.end 1
    geom.end select 1-28
    geom.end remove 7,8
```

支持以下形式：

```
charge[.<slot>] <N>
charge[.<slot>] add <N>
charge[.<slot>] subtract <N>
spin[.<slot>] <N>
spin[.<slot>] add <N>
spin[.<slot>] subtract <N>
geom[.<slot>] select <atom index list>
geom[.<slot>] remove <atom index list>
freeze <value> <atom index list>
oniom <layer> <atom index list>
fragment <id> <atom index list>
basisset <name>
```

作用域规则如下：

- 省略 **.<slot>** 时目标固定为主几何 **geom**，既有任务语义不变；辅助几何不会继承这些无作用域操作。
- **.main** 是主几何的输入别名，例如 **charge.main 0** 与 **charge 0** 等价；formatter 统一输出后者。
- 非主 slot 必须由同一任务的 **%source** 以 **geom.<slot>** 声明；slot 名大小写不敏感并规范化为小写。空 slot、非法名称或未声明目标会直接报错。
- **%mod** 只接受命名目标，不接受 **charge2**、**spin2**、**geom2** 等位置别名。数字别名仅用于 **[charge2]**、**[geom2]** 等运行期占位符。
- scoped 语义当前只覆盖 **charge**、**spin** 和 **geom select/remove**。**basisset**、**freeze**、**oniom**、**fragment** 及其他结构化操作继续作用于主几何或对应程序输入，不会复制到辅助几何。
- 每个 slot 的坐标过滤和电荷/自旋修改在该几何写入 staged XYZ、生成命名占位符之前完成。只修改辅助几何不会改变任务主几何的 **struc_id**。

几何索引采用相应来源几何的 **1-based** 编号，支持逗号与区间混写。例如，下面只保留主几何中的第 1、3、6 至 9 号原子：

```
%mod
  geom select 1,3,6-9
```

同一 slot 可以先保留一个片段，再剔除其中部分原子：

```
%mod
  geom.end select 1-20
  geom.end remove 7,8,15-17
```

任务 manifest 会在每个几何 binding 中记录规范化的 `mod_operations`。`comd-only` 只有在来源声明与 scoped `%mod` 均未改变时才复用已物化几何；声明变化时会拒绝旧 staged 文件，而不是静默生成过期命令。

2.2.15.1 冻结、分层与片段

`oniom` 和 `fragment` 当前只在 Gaussian 输入生成路径中生效，并写入 `.gjf` 的对应结构字段。`freeze` 同时服务于 Gaussian 与 XTB：在 Gaussian 中写回原子冻结列；在 XTB 中写入 xcontrol 约束，普通计算采用 fix 语义，分子动力学类输入采用 constrain 语义并默认 force constant 为 `1.0`。

这些指令与 `geom select/remove` 一样都采用**原始源结构的 1-based 编号**。如果某个原子先被 `geom remove` 剔除，那么后续 `freeze`、`oniom` 或 `fragment` 即使选中了它，最终写出的输入也会跳过该原子。对于 XTB，`freeze <value> <atom index list>` 中的 `<value>` 只作为语法占位保留，实际约束类型由 XTB 输入场景决定。

```
%mod
  geom select 1-20
  geom remove 7,8
  freeze -1 1-6
  oniom H 1-12
  oniom L 13-20
  fragment 1 1-12
  fragment 2 13-20
```

2.2.15.2 结构化基组

Gaussian 和 ORCA 共用 `%mod basisset` 的 assignment 语法：

```
basisset <name>
  [on=all|element:<comma-separated-elements>|atom:<index-list>]
  [role=orbital|aux-j|aux-jk|aux-c|cabs]
  [ecp=auto|none|required]
```

省略 `on=` 表示整个体系；省略 `role=` 表示 orbital basis；省略 `ecp=` 表示 `auto`。`on=atom:` 使用原始源结构的 **1-based** 编号，支持逗号与升序或降序区间，例如 `on=atom:1,4-7,12`。

同一任务可以声明多条 assignment。每个 role 的覆盖优先级固定为 `atom > element > all`，不依赖声明顺序；每个 role 最多有一个全体系默认值，同一 role、同一 specificity 的 selector 重叠时直接报错。

```
$g16
  %program gaussian
  %keywords ub3lyp opt
  %mod
    basisset ma-SVP
    basisset ma-TZVP on=element:Fe,Pt

$orca
  %program orca
  %keywords PBE0 RIJCOSX TightSCF
  %mod
    basisset def2-SVP
    basisset def2-TZVPP on=element:Fe
    basisset match role=aux-j
```

两种后端采用相同的 selector 和覆盖规则，但写入方式不同：

项目	Gaussian	ORCA
基组角色	仅 **orbital**	**orbital**、**aux-j**、**aux-jk**、**aux-c**、**cabs**
全体系 assignment	生成 general-basis 输入并调整 route 基组槽位	写入 **%basis** 的 **Basis**、**AuxJ**、**AuxJK**、**AuxC** 或 **CABS**
元素 assignment	按元素写 general-basis block	写入 **NewGTO** 或相应 **NewAux*GTO**
原子 assignment	使用 Gaussian 1-based center number	追加到 inline Cartesian 原子行，并启用 **PrintBasis**
ecp=	支持 **auto**、**none**、**required**	仅接受 **auto**；显式 ECP 应完全改用原生 **%basis**
辅助基组快捷名	不适用	**auto** / **autoaux** 使用 **AutoAux**；**match** 及其别名按 orbital basis 推导匹配项

Gaussian 的 assignment 必须覆盖所有原子；ORCA 的 orbital assignment 也应为需要覆盖的体系提供全体系或更具体的值。结构化 assignment 不得与手写 Gaussian basis/ECP section 或 ORCA **%basis** 混用。Gaussian 不接受辅助 role；ORCA 不允许把 **auto** 或 **match** 用作 orbital basis，且原子 assignment 要求 inline Cartesian geometry。

on=atom: 始终引用执行 **%mod geom select/remove** 前的原始结构编号。生成输入前，BaneTask 会把 selector 重映射到过滤后的有效结构；被删除的原子从 selector 中移除，selector 最终为空时任务准备失败。

Gaussian 可用名称和外部基组数据见Gaussian 基组 catalog；ORCA 的具体写出规则与辅助基组匹配规则见ORCA 的结构化基组。

2.2.16 %preprocess 与 %process

%preprocess 与 **%process** 分别用于描述主程序运行前后的命令序列。常见操作包括准备工作目录、复制辅助文件、执行后处理、归档工件以及提取自定义结果。

```
%preprocess
  mkdir plot
  cp ../opt/*.fchk .

%process
  publish --pin final.log
  result final_energy --exec grep "SCF Done" final.log
```

这两个块中的内容按“每行一条命令 DSL 或原样命令”的方式解释，并持续读取到下一个任务指令或任务块。块内的 **&INCLUDE** 会按 **preprocess** / **process** 片段在当前位置展开；片段中的空行与注释只用于排版。**%preprocess** 的内容会写入 **pre_comd**，**%process** 的内容会写入 **comd**；当结果快照导出链路启用时，系统还会在 **comd** 末尾自动追加 **btproc result** 调用。

命令 DSL 的详细规则见命令 DSL章节。

2.2.17 %meta

%meta 用于补充任务级结构标识信息。当前可用字段较少，主要用于把与当前任务强相关、且需要进入结果链路的标识单独抽出。

%meta 支持的字段为：

```
%meta
  strucid S000123
```

该字段在结果快照导出启用时会写入任务目录中的 **.bane.taskmeta.json**，在结果记录链路启用时会同步写入记录中的 **strucId**，同时也会导出到命令脚本环境变量 **BANETASK_STRUCID**。一般性的筛选标签属于 YAML 头部普通元数据，不属于 **%meta** 的主要用途。

2.2.18 无输入任务的行为

并非所有任务块都会生成新的程序输入文件。默认情况下，只有在 **%source** 可解析且当前任务具备可供对应程序消费的输入描述时，BaneTask 才会视其为一个需要正式生成输入的任务。

BaneTask 默认只在同时具备以下条件时生成程序输入文件：

- **%source** 可解析
- 当前任务具备可供所选后端消费的输入描述（例如 Gaussian / ORCA / GAMESS / AMESP / MOPAC / BDF / MRCC / MAPLE 的 **%keywords** 或 BDF 的纯 **%extrainput**，FCclasses 的结构化 **%source**，XTB 的 **%args**，**other** 的 **%template** / **%param**，或 **script** 的 **%runner** / **%body**）
- **%control externalinp** 为 **false** 或未设置
- 当前任务不是 **%interface provider**

若上述任一条件缺失，则当前任务通常会进入“由外部流程准备输入或仅刷新脚本”的路径。在这种情况下，若存在 **%preprocess**、**%process** 或 **autorun: true**，系统仍会刷新命令脚本；workflow 也仍会按照当前目录中的实际输入文件匹配规则尝试生成。

另外，若显式设置 **%control externalinp true**，语义会进一步变成：**输入文件由外部**

脚本准备，workflow 主运行命令照常生成。也就是说，外部脚本写出正确文件名的输入文件后，主程序执行仍进入 BaneTask workflow。`%interface provider` 是例外：provider 只作为接口描述被 consumer 读取，不会单独生成 workflow。

3 量化程序后端

程序后端负责把同一套任务对象转换为具体程序输入、运行配置和结果入口。每节按“字段与生成物、最小任务、输入规则、运行与结果、失败条件”组织。

3.1 程序能力总表

程序名	主输入	原生结构化输入	batch	interface	标准结果快照	结果侧重点
`gaussian`	`.gjf`	route、附加段、Link1	是	consume / provide	是	几何、能量、振动、热化学、轨道、激发态
`orca`	`.inp`	`!` 行与 `%...end` 块	是	consume / provide	是	几何、能量、梯度、振动、轨道、激发态
`gamess`	`.inp`	`$SECTION ... $END`	是	provide	是	几何、能量、轨道、梯度、振动
`amesp`	`.aip`	`%`、`!`、`>...end`	是	consume / provide	是	几何、优化、梯度、Hessian、振动、热化学、激发态
`mopac`	`.mop`	首行 token 与坐标	是	否	是	几何、能量、形成焓及 `.aux` 性质
`bdf`	`.inp`	easy、模块、Flat DSL	是	否	是	取决于 BDF 输出解析能力
`fcclasses`	`fcc.inp` + plan	flexible input 与结构化资源	计划驱动	否	按作业	光谱、量子产率、态与耦合资源
`mrcc`	`MINP`	MINP 字段	是	否	按解析能力	高级电子结构结果
`maple`	`.inp`	header 与坐标后命令	是	否	按解析能力	机器学习势能计算结果
`xtb`	`.xyz` + `.xcontrol`	命令参数与 xcontrol	是	provide	显式收集	几何、能量、extxyz 原子属性
`script`	`.sh` / `.bat` + payload	脚本文本	是	否	用户发布	任意脚本输出
`other`	模板指定	模板文本	取决于模板	否	用户发布	取决于模板与 Process DSL

程序名不区分大小写。任务文件应使用表中的规范名称；解析器接受的别名见第 6 章。

3.2 Gaussian

Gaussian 后端生成托管的 Gaussian 输入、检查点后处理命令和可选 External 接口脚本。

3.2.1 任务字段与生成物

项目	规则
`%program`	`gaussian`
主输入	`<task>_<origin>.gjf`
环境配置	`gaussian.conf`
主要任务字段	`%source`、`%keywords`、`%extrainput`、`%control`、`%interface`、`%mod`
结果入口	Gaussian 输出；可附带 `.chk`、`.fchk` 与结果快照

3.2.2 最小任务

```
$sp
  %source origin.xyz
  %program gaussian
  %control totcore 8
  %control totmem 16000
  %keywords B3LYP/def2SVP SP
```

3.2.3 `%keywords` 与 `%extrainput`

`%keywords` 是 Gaussian route 内容，不含开头的 `#`。`%control verbosity` 选择 `#`、`#p` 或 `#t`：

```
%keywords "opt freq B3LYP/6-31G(d)"
```

`%extrainput` 写在电荷、多重度和坐标段之后，适合附加基组、ECP、溶剂或 Link1 内容。可以使用 heredoc，也可以在空块中并入文本片段：

```
%extrainput << EOF
%chk extra.chk

--Link1--
%oldchk extra.chk
#p B3LYP/6-31G(d) SP Geom=AllCheck Guess=Read
EOF
```

```
%extrainput
  &INCLUDE "inputs/gaussian-extra.inp"
```

heredoc 体按原文写入；其中的 `&INCLUDE` 不再解释。空块中的 `&INCLUDE` 按文本片段读取。`%extrainputs` 是 `%extrainput` 的等价输入别名。

3.2.4 输入生成规则

Gaussian 输入依次写入资源指令、route、标题、电荷与多重度、几何、附加输入和 Link1 内容，并生成 `<task>_<origin>.gjf`。

- `%control totmem` 与 `totcore` 分别写入内存和核数指令。
- `%source` 与 `%mod charge/spin/geom` 决定最终电荷、自旋多重度和坐标。
- `%control guess` 在可解析时写入 `oldchk=../<guessTask>/<guessTask>_<origin>.chk`；route 未给出 `guess=...` 时自动追加 `guess=read`。
- `%control geom check` 会在需要时补写 `geom=check`，并通过同一检查点来源复用上游几何。
- `%interface consume <provider>` 会生成 `qc2g16_external` 配置和 External 启动脚本，并替换 `[external]`。provider 可为 Gaussian、XTB、ORCA、GAMESS 或 AMESP。
- `%mod freeze/oniom/fragment` 写回原子冻结列、ONIOM 层和 `Fragment=...` 标记。
- 常见 route 拼写变体会做有限规范化，例如 `m06-2x`、`wb97x-d`、`def2-svp`。

3.2.4.1 结构化基组生成

`%mod basisset` 只接受 `role=orbital`。全体系、元素和原子 assignment 按结构化基组 的共同规则解析；存在原子 selector 时，general-basis block 使用 Gaussian 的 1-based center number，因此同一元素的不同原子可以使用不同基组。

任一 assignment 需要 ECP 时，route 使用 `genecp`；否则使用 `gen`。`%mod basisset` 的优先级高于 `%keywords` 中的普通 Gaussian 基组，例如 route 中的 `b3lyp/6-31g(d)` 会替换为 `b3lyp/gen` 或 `b3lyp/genecp`。`%control geom check` 不写坐标，但仍可从结构模型取得元素并生成基组段。

已有 `gen` / `genecp`、手写 basis/ECP section、ONIOM、双层方法、density-basis 结构或隐含/固定基组 procedure 与结构化 assignment 冲突时，输入生成失败。

3.2.4.2 Gaussian 基组 catalog

Gaussian 外部基组由随 BaneTask 安装的基组 catalog 登记。catalog 条目提供规范名称、可接受别名、Gaussian basis 资源和可选 companion ECP。它有两种入口：

- 在 `%mod basisset` 中声明 assignment；
- 直接把 catalog 名称写入 Gaussian route 的方法/基组槽位。

当前可用于 Gaussian 的规范名称如下：

规范名称	常用别名	ECP 行为
`def2-mTZVP`	`def2-mTZVP`	对 catalog 指定的重元素范围自动使用 def2 ECP
`ma-SV(P)`	`ma-def2-SV(P)`	对 catalog 指定的重元素范围自动使用 def2 ECP
`ma-SVP`	`ma-def2-SVP`	对 catalog 指定的重元素范围自动使用 def2 ECP
`ma-TZVP(-f)`	`ma-def2-TZVP(-f)`	对 catalog 指定的重元素范围自动使用 def2 ECP
`ma-TZVP`	`ma-def2-TZVP`	对 catalog 指定的重元素范围自动使用 def2 ECP

规范名称	常用别名	ECP 行为
ma-TZVPP	ma-def2-TZVPP	对 catalog 指定的重元素范围自动使用 def2 ECP
ma-QZVP	ma-def2-QZVP	对 catalog 指定的重元素范围自动使用 def2 ECP
ma-QZVPP	ma-def2-QZVPP	对 catalog 指定的重元素范围自动使用 def2 ECP
pcseg-1	pcseg1、pcseg_1	不附加 ECP
maSVXP	ma-SVXP	对 catalog 指定的重元素范围自动使用 def2 ECP
def2SVSP	def2-SVSP	对 catalog 指定的重元素范围自动使用 def2 ECP
def2-TZVPPD	def2TZVPPD	对 catalog 指定的重元素范围自动使用 def2 ECP
def2-QZVPPD	def2QZVPPD	对 catalog 指定的重元素范围自动使用 def2 ECP

`ecp=auto` 使用条目声明的 ECP 策略；`ecp=none` 禁止当前 assignment 写 ECP；`ecp=required` 要求所选原子至少命中一个可用 ECP。名称不存在、基组资源缺少所需元素、ECP 策略不满足或体系存在未覆盖原子时，任务准备失败。

3.2.5 运行与结果

- `gaussian.conf` 的命令模板负责调用站点安装的 Gaussian；检查点存在时可由 workflow 调用 `formchk`。
- `btproc result` 从主输出提取几何、能量、振动、热化学、轨道和激发态等已支持属性。

3.2.6 失败条件与诊断

- 源几何不可解析、route 为空、guess 任务缺少可用检查点、External provider 不受支持或结构化基组无法解析时，准备阶段失败。
- 主程序非零退出、正常终止标记缺失或结果解析失败时，分别检查主输出、`run.log`、`bt.status` 与结果处理日志。

3.3 ORCA

ORCA 后端生成 `*.inp`，补齐并行块与每进程内存，并在主计算后处理 `.gbw`。

3.3.1 任务字段与生成物

项目	规则
%program	orca
主输入	<task>_<origin>.inp
环境配置	orca.conf
主要任务字段	%source、%keywords、%extrainput、%control、%interface、%mod
结果入口	ORCA 输出；可附带 .gbw 与 Molden 文件

3.3.2 最小任务

```
$opt
  %source origin.xyz
  %program orca
  %control totcore 8
  %control totmem 16000
  %keywords B3LYP def2-SVP Opt
```

3.3.3 %keywords 与 %extrainput

%keywords 形成 ORCA 的 ! 方法行:

```
%keywords "wB97M-V def2-TZVP def2/J RIJCOSX TightSCF"
```

%extrainput 用于 %scf、%geom、%basis 等 ORCA 原生块以及其他正文内容:

```
%extrainput << EOF
%scf
  MaxIter 300
end
EOF
```

heredoc 体按原文交给 ORCA 输入解析器；也可使用空块中的 &INCLUDE 并入文本片段。资源块和结构化 %basis 会在生成阶段检查，避免与自动资源设置或 %mod basisset 重复。

3.3.4 输入生成规则

ORCA 任务生成 <task>_<origin>.inp。主命令结束后，如果 <task>_<origin>.gbw 存在，workflow 自动执行 orca_2mkl <task>_<origin> -molden；转换步骤不覆盖主程序返回码。

- 默认资源来自 orca.conf 中的 CONF_CORES 与 CONF_MEMORY，%control totcore/totmem 可覆盖。
- %extrainput 未给出 %maxcore 时，生成器写入 totmem / totcore 得到的每进程内存。
- %extrainput 未给出 %pal nprocs ... end 时，生成器按 totcore 补写并行块。
- %source 与 %mod charge/spin/geom 决定最终几何和电子态。
- %interface 可把 ORCA 用作 Gaussian External provider 或 consumer，具体导数关键词由接口生成阶段补齐。

3.3.4.1 ORCA 的结构化基组

ORCA 支持 orbital、aux-j、aux-jk、aux-c 和 cabs。全体系 assignment 写入 %basis 的 Basis、AuxJ、AuxJK、AuxC 或 CABS；元素 assignment 写为 NewGTO、NewAuxJGTO、NewAuxJKGTO、NewAuxCGTO 或 NewCABSGTO，并自动启用 PrintBasis。

```
$orca
  %keywords PBE0 D4 RIJCOSX TightSCF
  %mod
    basisset def2-SVP
```

```
basisset def2-TZVPP on=element:Fe
basisset auto role=aux-j
```

原子 assignment 会把相应 `New*GTO ... end` 指令追加到 inline Cartesian 原子行，因此不能用于 `xyzfile` 等无法携带逐原子指令的几何形式。

辅助角色的 `auto` / `autoaux` 映射为 `AutoAux`。`match`、`matched`、`recommended` 和 `paired` 会根据同一 selector 最终生效的 orbital basis 推导匹配辅助基组：def2/ma-def2、cc/F12、x2c 和 SARC 等已知家族按明确规则选择 `/J`、`/JK`、`/C`、`-MP2Fit`、`-CABS` 或 `-OptRI`。无法可靠推导时直接报错，必须显式给出辅助基组或改用 `auto`。

已知别名会规范化为 ORCA 原生名称；其他非空名称按 ORCA 原生基组名原样写入。结构化 assignment 与手写 `%basis` 不允许混用。当前结构化路径仅接受 `ecp=auto`；需要显式 ECP 覆盖时，应完全使用 `%extrainput` 中的原生 `%basis`。

3.3.5 运行与结果

- `orca.conf` 负责调用 ORCA；主命令结束后，存在 `.gbw` 时可执行 `orca_2mkl ... -molden`。
- 结果解析覆盖几何、能量、梯度、频率、热化学、轨道和激发态等已实现属性。

3.3.6 失败条件与诊断

- 输入块不闭合、源几何不可解析、资源值无效、基组 assignment 冲突或辅助基组无法推导时，准备阶段失败。
- 转换步骤不覆盖主程序返回码；应先检查 ORCA 主输出，再检查 Molden 转换日志。

3.4 GAMESS

GAMESS 后端生成托管 `.inp`，也可在预处理阶段由波函数转换器创建目标输入。

3.4.1 任务字段与生成物

项目	规则
`%program`	`gamess`
主输入	`<task>_<origin>.inp`
环境配置	`gamess.conf`
主要任务字段	`%source`、`%keywords`、`%extrainput`、`%control guess_converter`、`%mod`
结果入口	GAMESS 输出与结果快照

3.4.2 最小任务

```
$sp
  %source origin.xyz
  %program gamess
  %keywords $CONTRL SCFTYP=RHF RUNTYP=ENERGY $END
  %extrainput
$BASIS GBASIS=N31 NGAUSS=6 $END
```

3.4.3 输入生成规则

- `%keywords` 与 `%extrainput` 合并为 GAMESS section 文本。
- 托管生成时，最终电荷、多重度和几何写入 `$CONTRL` 与 `$DATA`。
- `guess_converter` 可从受支持的 Gaussian 或 ORCA 波函数准备 GAMESS 输入。
- `$DATA` 和 `$VEC` 不作为普通关键词补丁写入转换器生成的输入。

GAMESS 任务会生成标准 `.inp` 文件，并复用 `%mod` 处理后的几何、电荷与自旋。GAMESS 的初猜数据通常嵌入输入文件本身；当目标任务配置了受支持的跨程序 `guess_converter` 时，BaneTask 会在预处理阶段由转换器生成目标 `.inp`，再执行 GAMESS 主命令。

GAMESS 任务会生成或声明：

- `<task>_<origin>.inp`

生成规则如下：

- 未启用跨程序 `guess_converter` 时，`%keywords` 与 `%extrainput` 会先合并为一份连续的 GAMESS section 文本，再交给输入桥接层规范化输出。
- 启用 Gaussian 到 GAMESS 或 ORCA 到 GAMESS 的 `guess_converter` 时，目标任务会按外部输入处理，`.inp` 由 `pre_comd` 中的转换命令创建。
- 转换器生成 GAMESS 输入后，`%keywords` 与 `%extrainput` 中可解析的普通 GAMESS 关键词会在预处理阶段合并到目标 `.inp`；`$DATA` 与 `$VEC` 不作为普通关键词补丁写入。
- `%mod charge/spin` 与 `%mod geom` 只作用于 BaneTask 托管生成的 GAMESS 输入；转换器生成的外部输入由上游波函数文件与转换工具决定。
- 托管生成时，`%source` 应用 `%mod charge/spin` 后的最终电荷和自旋多重度会写入 `$CONTRL ICHARG/MULT`，覆盖 `%keywords` 或 `%extrainput` 中的同名值。
- `$CONTRL SCFTYP` 未设置或取值为 `RHF` / `UHF` 时，会按最终自旋多重度自动选择：单重态使用 `RHF`，其他多重度使用 `UHF`。`ROHF`、`GVB`、`MCSCF` 等专用类型保持原值。
- `%control totcore` / `totmem` 主要影响 workflow 资源层；GAMESS 输入正文由托管输入生成器或转换器生成的 `.inp` 决定。

Gaussian 到 GAMESS 与 ORCA 到 GAMESS 的跨程序初猜转换见 `%control guess_converter`。

3.4.4 运行与结果

- `gamess.conf` 定义站点 GAMESS 启动命令和 scratch 环境。
- 结果解析读取主输出中已支持的几何、能量、轨道、梯度、频率与性质。

3.4.5 失败条件与诊断

- GAMESS section 语法无效、转换器来源不完整或转换命令失败时，任务不会进入主计算。

- 转换器路径应同时检查 `pre_comd`、转换日志和最终 `.inp`。

3.5 AMESP

AMESP 后端生成托管 `.aip` 输入。任务字段先解析为结构化 AMESP 文档，再由 schema 校验和渲染器写出全局指令、方法行、section 与最终坐标。

3.5.1 任务字段与生成物

项目	规则
`%program`	`amesp`
主输入	`<task>_<origin>.aip`；批量帧各自使用 `.aip`
环境配置	`amesp.conf`（默认输出后缀 `.aop`）
主要任务字段	`%source`、`%keywords`、`%extrainput`、`%control`、`%interface`、`%mod`、`%batch`
结果入口	AMESP `.aop`；可解析元数据、几何、优化、梯度、Hessian、振动、热化学与激发态

3.5.2 最小任务

```
$force
  %source origin.xyz
  %program amesp
  %control totcore 8
  %control totmem 16000
  %keywords UB3LYP 6-31G* FORCE
  %extrainput
>scf
  maxiter 200
end
```

3.5.3 输入生成规则

- `%keywords` 按空白拆分并形成 `!` 方法行。schema 登记的 token 会归一化；语法完整的原生 token 保留给 AMESP 检查。
- `%extrainput` 解析为 AMESP 全局指令、方法行和 `>section ... end`。解析失败时不会回退为不经检查的文本。
- `%control totcore` 映射为 `% npara`；`totmem` 映射为 `% maxcore`。设置核数时，`maxcore` 使用每核内存。
- `%source` 与 `%mod charge/spin/geom` 产生的最终结构覆盖附加输入中的 `>xyz`，保证几何、电荷和多重度只有一个事实来源。
- 普通任务写出 `<task>_<origin>.aip`；`%batch` 为每帧生成独立输入，批量启动器的汇总输出为 `run.aop`。

- `%interface consume <provider>` 可把 AMESP 作为 Gaussian External 消费端。导数请求标记为空、`force` 或 `numfreq`，分别对应能量、梯度或数值频率请求。

3.5.4 运行与结果

- `amesp.conf` 的默认命令形式为 `amesp "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}" 2>&1`，默认输入/输出后缀为 `aip`/`aop`。
- 该后端已接入标准结果快照链路。`btproc result` 可从 `.aop` 生成规范结果文档、KV 快照、最终几何和数据库记录。
- 解析器分别判断正常终止和结果有效性；有效结果至少需要可用几何。正常终止标记缺失不会被几何存在所掩盖。

3.5.5 失败条件与诊断

- 方法 token、section 字段、范围或枚举值无效时，先检查本节 schema 表，再检查 AMESP 主输出给出的程序级诊断。
- 主计算失败时检查 `.aop`、`run.log`、`bt.status` 和任务目录中的 manifest；批量任务还应逐帧检查 `.batch` 状态。

3.6 MOPAC

MOPAC 后端生成 `.mop` 输入，并保留 MOPAC 自行创建 `.out` 与 `.aux` 的同 stem 文件约定。

3.6.1 任务字段与生成物

项目	规则
`%program`	`mopac`
主输入	`<task>_<origin>.mop`
环境配置	`mopac.conf`
主要任务字段	`%source`、`%keywords`、`%extrainput`、`%control totcore`、`%mod`、`%batch`
结果入口	原生 `.out`，可合并同 stem `.aux`

3.6.2 最小任务

```
$opt
  %source origin.xyz
  %program mopac
  %control totcore 4
  %keywords PM7 OPT AUX
```

3.6.3 输入生成规则

- `%keywords` 与 `%extrainput` 按 MOPAC 首行 token 规则合并。
- 最终电荷、多重度和坐标来自 `%source` 与 `%mod`；`totcore` 映射为 `THREADS`。
- MOPAC 自行创建 `.out` 与 `.aux`，运行模板不得把标准输出重定向到主 `.out`。

MOPAC 任务生成托管 `.mop` 输入，并把输入与输出的同 stem 约定写入 `task.manifest.json`。任务关键字、电子态、坐标、资源和输出选项会在生成阶段统一校验和写入。

MOPAC 任务会生成或约定：

- `<task>_<origin>.mop`：主输入。
- `<task>_<origin>.out`：MOPAC 原生主输出。
- `<task>_<origin>.aux`：启用 `AUX` 时由 MOPAC 原生生成的辅助结果。
- `task.manifest.json`：记录 method、task、threads、AUX 设置和 `primary_output`。

生成与执行规则如下：

- `%keywords` 与 `%extrainput` 先按 MOPAC 第一行 token 规则合并；括号内空格和引号内容不会被错误拆分。
- 支持 AM1、INDO、MINDO3、MNDO、MNDOD / MNDO-D、PM3、PM6 系列、PM7 / PM7-TS 与 RM1 等方法关键字；其他未被托管字段识别的 token 会保留在 MOPAC 输入第一行。
- `%source` / `%mod` 处理后的坐标、电荷和多重度是电子态事实来源；`%control totcore` 映射为 `THREADS`。
- 默认 `mopac.conf` 使用 `SUFFIX=mop`、`OUTPUT_SUFFIX=out` 与 `mopac "${ACTUAL_INPUT_FILE}"`。因为 MOPAC 自己创建 `.out` / `.aux`，命令模板不得再把标准输出重定向到 `${ACTUAL_OUTPUT_FILE}`。
- 该后端已接入标准结果快照链路。`btproc result` 读取 `.out` 时会自动合并旁边同 stem 的 `.aux`，可生成规范 QC 文档、结果别名、最终 XYZ 和数据库记录；形成焓结果别名为 `formation_enthalpy`。
- `%batch` 的每帧输入和输出保持同 stem,例如 `scan_input.mop`、`scan_input.out`、`scan_input.aux`。启动器标准输出/标准错误单独写入 `run.log`，避免与 MOPAC 原生 `.out` 争用；成功帧可通过 `btproc fork collect mopac` 汇总到 `final/optimized_traj.xyz`。

示例：

```
$mopac_freq
  %program mopac
  %source opt
  %keywords "PM7 FORCE AUX(PRECISION=9,XP,XS,XW)"
  %control
    totcore 4
```

3.6.4 运行与结果

- `btproc result` 读取 `.out` 时自动合并同 stem `.aux`。
- 批量任务保持每帧 `.mop`、`.out` 与 `.aux` 同 stem；启动器日志写入独立 `run.log`。

3.6.5 失败条件与诊断

- 首行 token、电子态或坐标无效时，输入生成阶段失败。
- 结果缺失时同时检查 `.out`、`.aux` 与命令模板是否错误重定向了标准输出。

3.7 BDF

BDF 后端支持 easy、advanced 和 mixed 三种输入模式，并将 Flat DSL 或官方模块文本归一化为结构化 BDF 文档。

3.7.1 任务字段与生成物

项目	规则
`%program`	`bdf`
主输入	`<task>_<origin>.inp`
环境配置	`bdf.conf`
主要任务字段	`%source`、`%keywords`、`%extrainput`、`%mod`
结果入口	BDF 输出与标准结果快照

3.7.2 最小任务

```
$sp
  %source origin.xyz
  %program bdf
  %keywords B3LYP def2-SVP
  %extrainput
$scf
maxit
  200
$end
```

3.7.3 输入生成规则

- `%keywords` 提供 easy token。
- `%extrainput` 可使用官方 `$module ... $end` 或 BDF Flat DSL。
- easy 与模块同时存在时生成 mixed 输入；未知但语法完整的模块内容按 raw 记录保留。
- 最终几何、电荷与自旋会在渲染前覆盖文档中原有的几何、电荷与自旋。

BDF 任务生成托管 `.inp` 输入，并按结构化文档的实际内容选择 easy、advanced 或 mixed 模式：

- `%keywords` 提供 easy token；当它为空时，Flat DSL 的 `easy(...)` / `route(...)` 也可提供 easy token。
- 官方 `$module ... $end` 或 Flat DSL 的普通模块调用提供 advanced 部分。

- 只有 easy token 时生成 easy input，只有模块时生成 advanced input，两者同时存在时生成 mixed input。
- `%extrainput` 自动识别官方 BDF 文本或BDF Flat DSL，并把两种写法归一化为同一份结构化 BDF 文档。
- `%mod charge/spin/geom` 在结构化文档进入渲染前生效；托管任务中的 `%source` / `%mod geom` 会覆盖 Flat DSL 文档原有的几何设置。
- 未被内置 schema 收录的模块和关键词默认按 raw 内容保留；完整文档在写出前统一校验。
- 输出文件名为 `<task>_<origin>.inp`，workflow 默认读取 `bdf.conf`。
- BDF 已接入标准 `btproc result extract` 快照链路；实际属性覆盖取决于当前安装版本支持的 BDF 输出解析能力。

3.7.3.1 BDF Flat DSL

`%extrainput` 内也可以写 BaneTask 约定的 BDF Flat DSL。Flat DSL 用函数式的紧凑写法表达 BDF 模块、关键词和特殊记录。以下示例会先归一化为结构化 BDF 文档，再渲染为官方 `$compass`、`$scf`、`$tddft`、`$resp` 等块：

```
$bdf_nac
  %program bdf
  %source origin
  %extrainput << EOF
compass(Title="C10H8",Basis=Def2-SVP,Skeleton,Group=C(1))
xuanyuan(Direct)
scf(RKS,Charge=0,SpinMulti=1,DFT=GB3LYP,D3,Molden)
tddft(Imethod=1,Isf=0,Idiag=1,Iroot=6,Istore=1)
resp(Quad,FNAC,Norder=1,Method=2,Nfiles=1,Double,
     Pairs=<1|1 1 1 1 1 2>)
EOF
```

`%extrainput` 默认自动区分 BDF Flat DSL 和官方输入。跳过空白、`#` 注释和 `//` 注释后：

- 第一个标识符后是 `(` 时，按 Flat DSL 解析，例如 `scf(...)`。
- 第一个标识符后是 `<` 时，只有 `shell`、`database`、`raw`、`comment`、`geometry` 被识别为 Flat DSL 特殊块。
- 其他内容按官方 BDF 文本解析；以 `$module` 开始的原生输入不会被误判为 Flat DSL。

模块调用的基本形式为：

```
module(Flag, Name=value, Structured=<line1|line2>)
```

形式	含义	示例
`module(...)`	一个 BDF 模块；模块可重复，原始 statement 顺序会保留	`scf(RKS)`
`Flag`	无值关键词或开关	`Molden`
`Name=value`	命名标量参数	`Charge=0`
`Name="value"` / `Name='value'`	带空格、逗号或引号的标量	`Title="S1, bright state"`

形式	含义	示例
`Name=<...>`	结构化值；最外层 `|` 转为物理换行	`Pairs=<1|1 1 1 1 1 2>`

参数之间用逗号分隔，右括号前允许尾逗号。statement 可用空白或换行分隔，顶层分号 `;` 可选。标识符以字母或 `_` 开始，后续还可包含数字、`-` 与 `.`。裸值允许嵌套圆括号，因此 `Group=C(1)` 不需要额外引号。

重复模块和重复字段不会被折叠。例如：

```
scf(RHF,SadAvgPartFor=6,SadAvgPartFor=8)
shell<cp $BDF_WORKDIR/$BDFTASK.scforb $BDF_TMPDIR/$BDFTASK.inporb>
scf(RKS,Guess=Readmo)
```

会保留两个 `scf` 模块、两个 `SadAvgPartFor` 条目以及中间 shell record 的顺序。

`<...>` 内最外层的 `|` 表示一条新的物理行；嵌套的 `<...>` 会原样保留并交给 schema 解释。常见形式包括：

```
compass(Geometry=<C 0 0 0|H 0 0 1>,Basis=Def2-SVP)
scf(ThreshConv=<1.d-6 1.d-4>)
bdfopt(Dimer-Block=<NoInterpolation|Delta=0.02|Crude>)
resp(Double,Pairs=<2|1 1 1 1 1 2|1 1 1 1 1 3>)
```

已知字段的具体结构类型由 BDF schema 决定，例如 terminated block、tuple 或 nested block。未知模块和关键词在默认任务路径中按 raw 内容保留；完整结构化文档在写出前统一校验。语法错误会报告行号和列号。

特殊 statement

写法	作用	约束
`easy(...)` / `route(...)`	写入 easy-input token	参数只能是标量
`script("name.bdf")`	设置独立 Flat DSL 文档的 script name	必须且只能有一个标量参数
`geometry<...>`	写入独立 Flat DSL 文档的笛卡尔几何	不能与 `compass(Geometry=<...>)` 同时出现
`shell<...>`	写入 `%` shell record；缺少 `%` 时自动补上	多行按 `|` 分隔
`database<...>`	写入 `&database ... &end`	多行按 `|` 分隔
`raw<...>`	原样写入 record	多行按 `|` 分隔
`comment<...>`	写入注释；无 `#` 或 `*` 前缀时自动补 `#`	可与普通 `#` / `//` 注释混用

在普通 BaneTask 托管任务中，任务的 `%source`、`%mod geom` 以及生成的 `<task>_<origin>.inp` 文件名是最终权威值：渲染前会重新设置几何与 script name。因此 `geometry<...>` 和

`script(...)` 主要用于独立 Flat DSL 文档，不应用来覆盖任务块的 source 或文件名。类似地，显式 `%keywords` 会作为最终 easy token 集合；只有 `%keywords` 为空时，Flat DSL 中的 `easy(...)` / `route(...)` 才直接提供 easy-input token。

转义与注释

- 引号字符串支持 `\\`、`\'`、`\"`。
- 结构化值支持 `\\`、`\|`、`\<`、`\>`；被转义的 `|` 不换行。
- 未识别的转义序列会保留反斜杠。
- `#` 与 `//` 可作为行注释，并默认保留到结构化文档。

输入模式

结构化文档按实际内容确定模式，而不只看 `%extrainput` 是否为空：

- 仅有 easy token：easy。
- 仅有模块：advanced。
- easy token 与模块同时存在：mixed。

`%keywords` 非空时提供 easy token；Flat DSL 的 `easy(...)` / `route(...)` 也可在 `%keywords` 为空时提供 easy token。官方 `$module ... $end` 或 Flat DSL 模块提供 advanced 部分。

处理顺序如下：

```
BaneTask 任务 DSL / %extrainput
  -> 识别 Flat DSL 或官方 BDF 文本
  -> 归一化并校验结构化文档
  -> 应用托管几何、电子态和文件名
  -> 渲染官方 BDF .inp
```

生成文件为 `<task>_<origin>.inp`。纯 advanced input 会自动补 `$compass` 几何，并在 `$scf` 中写入 charge/spinmulti；mixed input 会同步 easy 与 `$scf` 中的电子态，避免 advanced 部分覆盖 easy 设定。workflow 默认读取 `bdf.conf`，并使用 `${BDFHOME}/sbin/bdfdrv.py -r` 运行。

当前 BDF 后端已启用统一结果快照，workflow 会接入标准 `btproc result extract` 链路。具体可查询属性仍取决于当前安装版本支持的 BDF 输出解析能力和实际输出内容；某项属性不可解析时应按普通“结果不可用”处理，而不是假定所有 BDF 输出都具有完整字段。

3.7.4 运行与结果

- `bdf.conf` 定义站点启动命令；该后端已启用结果快照。
- 可查询属性由当前安装版本实际支持并成功提取的内容决定。

3.7.5 失败条件与诊断

- Flat DSL、官方模块边界或结构化文档校验失败时，准备阶段终止。
- 结果属性缺失时先确认 BDF 输出包含相应分析，再检查属性注册表中的 requirement。

3.8 FCclasses

FCclasses 后端从结构化来源恢复态、几何、振动、偶极和耦合资源，生成一个或多个独立 `fcc.inp` 作业和执行计划。

3.8.1 任务字段与生成物

项目	规则
`%program`	`fcclasses`
主输入	每个计划项目录中的 `fcc.inp`；总计划为 `fcclasses.plan.json`
环境配置	`fcclasses.conf` 与可选 `BANETASK_FCCLASSES_INPUT_TEMPLATE`
主要任务字段	`%source` 结构化资源、`%keywords`、`%extrainput`、`%batch`
结果入口	FCclasses 输出、计划状态和可选汇总结果

3.8.2 最小任务

```
$em
  %program fcclasses
  %source
    em(cart):
      state1: s0
      state2: s1
  %keywords spectrum em
```

3.8.3 输入生成规则

- `%source` 必须给出 FCclasses 可识别的结构化资源，而不是普通单一几何来源。
- 规划器按态对、坐标类型、property 和批次生成计划项；执行器只运行状态为 `ready` 的项目。
- 每个 `fcc.inp` 依次合并站点模板、恢复的资源字段、当前 `PROPERTY`/`COORDS` 与 `%extrainput`。
- `%keywords` 表达规划意图；FCclasses3 原生细节应放在模板或 `%extrainput`。

FCclasses 后端从结构化 `%source` 恢复上游资源并生成一个或多个独立作业。它不先解析单个主几何，也不扫描项目中的全部 `*.inp`；执行器只运行 `fcclasses.plan.json` 中状态为 `ready` 的 `fcc.inp`。

3.8.3.1 输入合并与配置

每个 `fcc.inp` 按以下顺序组成：

1. `BANETASK_FCCLASSES_INPUT_TEMPLATE` 指定的一个常规 flexible input；未配置时从 `$$$` 开始。
2. BaneTask 为当前态对准备的 `STATE1_FILE`、`STATE2_FILE`、偶极/NAC/coupling 字段。
3. 当前作业的 `PROPERTY` 与 `COORDS`。
4. `%extrainput` 原文，作为最后覆盖层。

`%keywords` 是规划层，不复制 FCclasses3 的整套输入语法；未写出的原生选项直接采用 FCclasses3 自身默认值。因而常见基线 `MODEL=AH`、`METHOD=TD`、`DIPOLE=FC`、`TEMP=0`、`BROADFUN=GAU`、`HWHM=0.01`、`NORMALMODES=COMPUTE` 通常无需在每个任务中重复。模板或 `%extrainput` 只应保留确需指定的非默认设置、需要测试的原生组合，或规划层未直接表达的 property，例如：

```
DIPOLE   = HTf
BROADFUN = VOI
HWHM     = 0.02
CLASSIC  = NO
```

最终输入不会经过要求认识全部 FCclasses 关键字的白名单校验。无法识别的赋值和文本保持原样；是否为有效 FCclasses 输入，以实际运行的 `fcclasses3` 为准。已知文件赋值必须是安全相对路径，并从 `asset`、case 输入目录、任务目录或模板目录中唯一解析后复制到作业目录。

3.8.3.2 state、偶极和耦合资源

state 文件按以下规则准备：

- 优先使用上游已有的唯一 `.fcc`，或 `file=` 明确指定的 `.fcc`。
- 对 `.fchk` / `.fch`，优先使用内建转换，直接读取 geometry、energy、gradient 和 Hessian 并写出 `.fcc`。
- 对其他受支持输出，或原生 fchk 转换失败时，才回退到 `BANETASK_FCCLASSES_GEN_STATE`；外部调用固定写为 `gen_fcc_state -i <absolute-input> -o <absolute-output>`。

电/磁跃迁矩优先使用 `eldip` / `magdip` 绑定的现有文件；否则对 formatted checkpoint 优先使用内建 `ETran` 转换，分别写出 electric 或 magnetic sidecar 和可用的 Cartesian 导数。Gaussian response 数据以 `S0<->Sn` 保存，系统会把 `S1->S0` 这类反向请求规范化，不再要求任务内 `%process`。内建转换失败或来源不是 fchk 时，才回退到 `gen_fcc_dipfile`；外部 fallback 同样规范化 root，并显式传递 `-Si`、`-Sf`、`-oe`/`-om`。未显式绑定 `eldip` 时，自动模式会尝试从两个 state 来源任务中寻找唯一 formatted checkpoint。

NAC 优先使用现有 `nac` 文件。BDF NAC 输出由内置结果解析读取，默认采用最后一个 electron-translation-corrected 向量，`variant=raw` 可选择 raw 向量；Gaussian formatted checkpoint 优先通过内建解析读取 `Nonadiabatic coupling`，失败时回退到 `gen_fcc_dipfile -nac -on <output>`。由 `%program other` + BDF 模板生成的 NAC 任务也按模板的最终 `prog=bdf` 识别。

SOC 输出按 singlet/triplet root 或 `pair=` 匹配，`component=` 控制优先采用 `M_S` 或 Cartesian XYZ 分量。匹配到的 SOC 模长从 `cm-1` 转为 atomic unit，并写为：

```
PROPERTY = NR0
NR0_COUPL = <value-in-au>
```

有 `nrcoup` 文件时，`isc` 优先生成 `PROPERTY=NRSC` 与 `NRCOUP_FILE`。BaneTask 默认不从 SOC 输出写入 `DE`；未由 `%extrainput` 明确覆盖时，FCclasses3 从两个 state 文件的 `ENER` 数据取得能量差。

规划层意图只自动规划 OPA、EMI、IC、NRSC 和 NR0，不把 ECD、CPL、MCD 等 property 视为 `abs` / `em` 的隐式别名。高级用法可以在 `%extrainput` 中最终覆盖原生 `PROPERTY`，并通过

eldip、**magdip** 或 **asset** 提供该 property 所需文件；此时资源组合和原生选项是否合法由实际 FCclasses3 判断，BaneTask 只负责已知文件字段的安全 staging。

3.8.3.3 量子产率任务

量子产率统计应把同一个初态的辐射衰变、内转换和系间窜越放在同一个 **phi** 任务中。下面的任务把 **S0** 和 **T1** 声明为 **S1** 的衰变目标态，并分别为同自旋和异自旋通道提供所需资源：

```
$phi_s1
  %program fcclasses
  %source
    state1 s1_optfreq state=S1
    state2 s0_optfreq state=S0
    state2 t1_optfreq state=T1
    eldip s1_optfreq pair=S1,S0
    nac nac_s1_s0 pair=S1,S0
    soc soc_s1_t1 pair=S1,T1 component=prefer-ms
  %keywords "phi(both)"
```

该任务最多展开 **EMI-S1-S0-***、**IC-S1-S0-*** 和 **NR0-S1-T1-*** 三类速率作业。若 **S1,T1** 提供的是 scalar coupling 文件，则对应作业使用 **NRSC**。**CARTESIAN** 与 **INTERNAL** 分别统计，不会相加为一个结果。

3.8.3.4 目录、计划和执行结果

任务目录的主要结构为：

```
<task>/
  fcclasses.plan.json
  fcclasses.summary.json
  fcclasses.report.txt
  fcclasses.report.json
  fcclasses.report.tsv
  <task>_<origin>.fcplan
  resources/
  jobs/
    EMI-S1-S0-CART/
      fcc.inp
      state1.fcc
      state2.fcc
      eldips.dat
    IC-S1-S0-INT/
      fcc.inp
      state1.fcc
      state2.fcc
      nac.dat
```

fcclasses.plan.json 记录自动、显式或量子产率模式、态标签与来源、property、速率类型、坐标、coupling 来源、**ready** / **not_planned** / **deduplicated** 状态及原因。**<task>_<origin>.fcplan** 按计划逐个进入 ready 作业目录并运行：

```
fcclasses3 fcc.inp
```

可通过环境变量 **FCCLASSES3** 覆盖主程序可执行路径。FCclasses3 对 **fcc.inp** 自动写出的 **fcc.out** 会被规范化为 **fcclasses.out**；每个作业还分别保存 **fcclasses.stdout** 和 **fcclasses.stderr**。启动器生成 **fcclasses.summary.json**，记录每个 ready 作业的退出码和输出路径。

启动器随后生成三种任务级速率报告，并把文本报告打印到当前 FCclasses 任务日志：

- **fcclasses.report.txt**：便于直接阅读的通道表和分区速率汇总。汇总按 System、Rates、Derived quantities 和 Completeness 分区纵向排列，温度、速率、寿命和量子产率统一使用小数点后六位的科学计数法。
- **fcclasses.report.json**：保留每个计划条目的状态、速率、输入设置和结构化汇总。
- **fcclasses.report.tsv**：便于表格软件或统计脚本读取的汇总表；数值字段使用小数点后六位的科学计数法。

速率报告按初态和坐标分别汇总：

```
kr_total   = sum(EMI rates)
kIC_total  = sum(IC rates)
kISC_total = sum(NR0 and NRSC rates)
k_total    = kr_total + kIC_total + kISC_total
tau        = 1 / k_total
Phi_r      = kr_total / k_total
```

只有纳入统计的三类速率覆盖完整时，报告才给出 **Phi radiative**。显式任务只计算了部分通道，或 **phi** 任务因资源缺失留下 **not_planned** 通道时，该字段为 **N/A**；报告仍会给出 **Phi from available rates**，并把覆盖状态标为 **partial**。缺失或未请求的通道不会被当作零。对 **phi** 任务，如果声明的全部 **state2** 中没有某一类物理上适用的跃迁，例如未声明任何异自旋低能态，则该类速率在当前声明态集合内按零计入。

作业成功需要同时满足：主程序返回码为零、**fcclasses.out** 含有正常结束标志，并且速率 property 的输出中能提取对应速率。任一条件不满足时，报告记录失败状态，FCclasses 任务也返回非零。Windows 环境中的 **fcclasses.conf** 通过 Bash 执行该启动器，因此需要在 **PATH** 或 **ENV_SETUP** 中提供 Git Bash/MSYS2 Bash 与 FCclasses3。

case 或 project 级汇总会把各任务中的失败和不完整通道写入报告及 **failure_count**，但不会跨任务合并速率。完整量子产率应由一个 **phi** 任务在统一的初态、目标态集合和计算设置下生成，避免把不同模型、温度或来源的独立任务误加。已有任务报告中的失败不会中断其他收尾动作；汇总文件本身无法读取、解析或写出时才返回非零。任务级启动器仍采用上述严格成功判定。

3.8.4 运行与结果

- **fcclasses.plan.json** 是准备与执行之间的事实来源，包含作业目录、输入、状态和跳过原因。
- 量子产率和多作业汇总使用相应计划与 **btproc fcclasses report**。

3.8.5 失败条件与诊断

- 所需态、振动、偶极、NAC 或 coupling 资源缺失时，计划项标记为不可运行并给出原因。
- 诊断时先检查 **fcclasses.plan.json**，再检查各作业目录的 **fcc.inp**、程序输出和状态文件。

3.9 MRCC

MRCC 后端直接生成固定文件名 **MINP**，将行内 **key=value**、MINP 多行片段、几何和电子态合并。固定文件名是受管理输入本身，不再通过运行前复制或重命名得到。

3.9.1 任务字段与生成物

项目	规则
%program	**mrcc**
主输入	**MINP**
环境配置	**mrcc.conf**，使用 **INPUT_FILENAME=MINP**
主要任务字段	**%source**、**%keywords**、**%extrainput**、**%mod**、**%control**
结果入口	MRCC 输出；结果能力取决于解析器覆盖

3.9.2 最小任务

```
$sp
  %source origin.xyz
  %program mrcc
  %keywords calc=CCSD(T) basis=def2-SVP
```

3.9.3 输入生成规则

MRCC 任务会生成：

- **MINP**
- **task.manifest.json**

生成规则如下：

- **%keywords** 作为 MRCC 行内 **key=value** 覆写入口。
- **%extrainput** 作为 MRCC MINP 多行片段；生成器会把它与 **%keywords**、几何、电荷和自旋一并写入 **MINP**。
- **%mod charge/spin** 会在写入 **MINP** 前生效；**%mod geom** 会在写入几何前生效。
- **%control totcore** / **totmem** 主要影响 workflow 资源层。
- MRCC workflow 默认读取 **mrcc.conf**。该配置以 **INPUT_FILENAME=MINP** 精确检索输入，并直接在任务目录运行 **dmrcc**；不得同时再配置 **SUFFIX**。

示例：

```
$mrcc_sp
  %program mrcc
  %source origin
  %keywords "calc=CCSDT basis=cc-pVDZ"
  %extrainput << EOF
scftype=RHF
mem=8000MB
EOF
```

3.9.4 运行与结果

- `mrcc.conf` 决定环境、运行命令和输出位置；固定输入名由 `INPUT_FILENAME=MINP` 声明。
- workflow 名与默认主输出名分别为 `MINP.bwrk` 和 `MINP.<OUTPUT_SUFFIX>`；内置配置使用 `MINP.out`。
- manifest 直接记录输入 `MINP` 与执行事实；可解析结果由 MRCC 输出适配器决定。

3.9.5 失败条件与诊断

- 关键词格式、几何、`MINP` 写出或环境配置选择器错误时，任务不能正常启动。
- 检查 `MINP`、`pre_comd`/`comd`、manifest，以及 `mrcc.conf` 是否只配置了 `INPUT_FILENAME=MINP`。

3.10 MAPLE

MAPLE 后端生成托管 `.inp`，把 header command、坐标和坐标后的约束或后处理命令分区写出。

3.10.1 任务字段与生成物

项目	规则
`%program`	`maple`
主输入	`<task>_<origin>.inp`
环境配置	`maple.conf`
主要任务字段	`%source`、`%keywords`、`%extrainput`、`%mod`、`%control`
结果入口	MAPLE 输出；结果能力取决于解析器覆盖

3.10.2 最小任务

```
$opt
  %source origin.xyz
  %program maple
  %keywords model=ani2x device=cpu opt(method=lbfgs)
```

3.10.3 输入生成规则

- `%keywords` 写入 MAPLE header commands。
- `%extrainput` 中的 header 行放在坐标前；约束、扫描和后处理命令放在坐标后。
- 最终几何、电荷和自旋在渲染前写入。
- GPU 设备由 MAPLE header、启动命令或站点环境选择；`totcore`/`totmem` 只参与 workflow 资源管理。

MAPLE 任务生成托管 `.inp` 输入文件，并复用 `%mod` 处理后的几何、电荷与自旋。BaneTask 写出的文件名为 `<task>_<origin>.inp`，写出时会按 MAPLE 输入格式进行规范化。

MAPLE 任务会生成：

- `<task>_<origin>.inp`
- `task.manifest.json`

生成规则如下：

- `%keywords` 写 MAPLE header commands。既可以写紧凑形式 `model=ani2x device=cpu opt(method=lbfgs)`，也可以直接写 `#model = ani2x` 这类 MAPLE 原生命令。
- `%extrainput` 中以 `#` 开头或可识别为 header command 的行会并入坐标前的 header；其他行会写到坐标后，适合放置 `C`、`B`、`A`、`D`、`S`、`POST` 等 MAPLE 后处理/约束/扫描命令。
- `%mod charge/spin` 会在写入 MAPLE 输入前生效；`%mod geom` 会在写入几何前生效。
- `%control totcore` / `totmem` 主要影响 workflow 资源层；MAPLE 的运行命令由 `maple.conf` 决定。
- MAPLE workflow 默认读取 `maple.conf`，环境配置的 `SUFFIX` 通常应与托管输入后缀 `inp` 保持一致。

示例：

```
$maple_opt
  %program maple
  %source origin
  %keywords "model=ani2x device=cpu opt(method=lbfgs)"
  %extrainput << EOF
B 1 2
S 1 2 0.05 10
EOF
```

3.10.4 运行与结果

- `maple.conf` 决定程序路径和启动命令。
- 任务 manifest 保存输入与运行参数；标准结果入口取决于 MAPLE 输出解析能力。

3.10.5 失败条件与诊断

- header command、坐标后命令或几何格式无效时，生成器拒绝写出输入。
- 设备选择问题应检查 MAPLE 输入 header、站点启动脚本和程序输出，而不是添加未被 workflow 消费的配置键。

3.11 XTB

XTB 后端写出标准化 XYZ、命令行参数和可选 xcontrol sidecar。

3.11.1 任务字段与生成物

项目	规则
`%program`	`xtb`
主输入	`<task>_<origin>.xyz`；可选 `.xcontrol`
环境配置	`xtb.conf`
主要任务字段	`%source`、`%args`、`%extrainput`、`%mod freeze`、`%mod geom`
结果入口	XTB 输出；可由 Process DSL 或 extxyz 导入结果

3.11.2 最小任务

```
$opt
  %source origin.xyz
  %program xtb
  %args --gfn 2 --opt
```

3.11.3 输入生成规则

- `%args` 进入 manifest 并展开到 XTB 命令行。
- `%extrainput` 与冻结约束生成 `.xcontrol`；BaneTask 自动管理 `--input` 参数。
- `[xyz_file]`、`[charge]`、`[spin]` 和 `[uhf]` 在写出前替换。
- 冻结约束使用源结构的 1-based 原子编号，并跟随几何筛选更新。

XTB 任务从 `%source` 解析几何并写出 `*.xyz`；存在 `%extrainput` 或冻结约束时，同时写出 `*.xcontrol`。

XTB 任务会生成：

- `<task>_<origin>.xyz`
- `<task>_<origin>.xcontrol`（仅在存在 `%extrainput` 或 `%mod freeze` 约束时生成）
- `task.manifest.json`

生成规则如下：

- `%args` 写入 `task.manifest.json` 的程序参数，并在 workflow 主命令中作为 XTB 命令行参数使用。
- `%args` 与 xcontrol sidecar 正文在写入前会替换 `[xyz_file]`、`[charge]`、`[spin]`、`[uhf]`；其中 `[uhf] = [spin] - 1`。
- `RUN_CMD_TEMPLATE` 若包含 `${ARGS}`，workflow 会在该位置展开 `%args`；若 XTB 模板未显式包含 `${ARGS}`，系统会在重定向前自动注入这组参数。
- `%extrainput` 会写入 xcontrol sidecar 正文。

- `%mod freeze` 会转换为 xcontrol 约束；约束索引仍按原始源结构的 1-based 编号解释，并会跟随 `geom select/remove` 过滤。
- 当 xcontrol sidecar 被生成时，BaneTask 会管理 `--input <xcontrol>` 参数；用户不应在 `%args` 中再手写 `--input` 或 `--input=...`。
- XTB workflow 默认读取 `xtb.conf`。托管输入流程中，环境配置的 `SUFFIX` 应与生成文件后缀 `xyz` 保持一致。
- XTB 优化、频率快照需要通过 `%process` 中的 `result` / `dbcollect` 写入数据库，也可通过 `btproc result extxyz-kv` 从 extxyz 文件导出 per-atom 属性。

示例：

```
$xtb_opt
  %program xtb
  %source origin
  %args --gfn 2 --opt tight
  %mod
    freeze -1 1-4
  %extrainput << EOF
$wall
  potential=logfermi
EOF
```

3.11.4 运行与结果

- `xtb.conf` 的 `RUN_CMD_TEMPLATE` 应包含或允许自动注入 `${ARGS}`。
- 优化、频率和原子属性可通过 `%process result`、`dbcollect` 或 `btproc result extxyz-kv` 进入结果链路。

3.11.5 失败条件与诊断

- 用户重复提供 `--input`、xcontrol 生成失败或源几何不可用时，准备阶段报错。
- 结果缺失时检查主输出、extxyz 文件、`%process` 和数据库收集步骤。

3.12 script

`script` 后端生成 Bash 或 Cmd 启动器，并在需要时拆分 launcher 与 payload。

3.12.1 任务字段与生成物

项目	规则
`%program`	`script`
主输入	`<task>_<origin>.sh` 或 `.bat`；可选 `.payload.<ext>`
环境配置	由 `%runner`、`BANETASK_SHELL` 与站点 shell 环境决定
主要任务字段	`%runner`、`%body`、`%source`、`%batch`、`%preprocess`、`%process`
结果入口	脚本退出码、日志及用户显式发布的结果

3.12.2 最小任务

```
$analysis
  %program script
  %runner python3
  %body
from pathlib import Path
Path("done.txt").write_text("ok\n")
```

3.12.3 输入生成规则

- `%runner` 选择解释器；缺失时可从 `%keywords` 读取替代值，最终默认 `/bin/bash`。
- `%body` 是脚本正文。runner 与平台 shell 不匹配时生成 launcher + payload。
- 几何占位符仅在正文引用时导出并替换。
- `%batch` 为每帧建立独立工作目录、输入、日志和状态。

`script` 任务生成可执行启动器，并可携带独立 payload。workflow 仍按普通任务处理其依赖、资源、状态和前后处理。

script 任务会生成：

- Bash 启动器：`<task>_<origin>.sh`
- Cmd 启动器：`<task>_<origin>.bat`
- 可选 payload：`<task>_<origin>.payload.<ext>`
- 批处理内部工作区：`.batch/`（仅启用 `%batch` 时；包含 manifest、平台对应的 `run_batch.sh` 或 `run_batch.bat`，以及 frame_XXXXXX/ 子目录）

生成规则如下：

- `%runner` 是脚本 runner 的规范字段；若缺失，则读取 `%keywords` 中的替代值；两者都缺失时默认使用 `/bin/bash`。
- `%body` 是脚本正文。
- 运行时 shell 由 `BANETASK_SHELL` 决定；`USE_GITBASH` 是 Windows 别名。生成器据此选择 Bash 或 Cmd 启动器。
- 当 runner 与启动器不适合写成单文件时，系统会生成 launcher + payload。例如 Bash 启动器下的 Python runner 会生成 `.sh` launcher 和 `.payload.py`。
- 若脚本正文中出现 `[xyz_file]`、`[geom]`、`[charge]`、`[spin]` 或 `[uhf]`，生成器会按需导出标准化 XYZ、几何正文、电荷、自旋多重度与 `spin - 1` 值，并在写脚本前完成替换。由 `[xyz_file]` 生成的标准化 XYZ 会把第二行写成类 Gaussian 的 `< 电荷 > < 自旋多重度 >`，例如 `0 1`。
- `%batch` 会把多帧 XYZ 来源展开到 `.batch/frame_XXXXXX/` 子目录；Bash 每帧生成 `run.sh` 并由 `.batch/run_batch.sh` 调度，Windows Cmd 每帧生成 `run.bat` 并由 `.batch/run_batch.bat` 通过 lane worker 与独立 inner `cmd.exe` 调度。两种 shell 都保留独立 `input.xyz`、`run.log` 与 `status.json`。
- `%process` / `%preprocess` 仍然独立生成到 `comd` / `pre_comd`。

3.12.4 运行与结果

- 脚本主退出码决定 workflow 主状态；用户结果必须通过 Process DSL、文件或数据库命令显式发布。
- `%preprocess` 和 `%process` 仍分别写入 `pre_comd` 与 `comd`。

3.12.5 失败条件与诊断

- runner 不存在、payload 无法启动或正文返回非零状态时，检查启动器、payload、`run.log` 和平台 shell 选择。
- 跨平台任务应避免依赖未在目标环境安装的 shell 内建命令。

3.13 other

`other` 后端使用命名模板渲染非内建程序输入，并用 `prog` 参数选择对应运行配置。

3.13.1 任务字段与生成物

项目	规则
`%program`	`other`
主输入	`<task>_<origin>.<suffix>`
环境配置	`%param prog=<name>` 选择 `<name>.conf`
主要任务字段	`%template`、`%param`、`%source`、`%keywords`、`%extrainput`、`%mod`
结果入口	由模板、运行配置和显式 Process DSL 定义

3.13.2 最小任务

```
$job1
  %source origin.xyz
  %program other
  %template myprog
  %param prog=myprog suffix=inp
  %keywords method=B3LYP
```

3.13.3 输入生成规则

- `%template <name>` 选择 `<name>.<suffix>` 模板。
- `%param prog=<name>` 选择 `<name>.conf`；`suffix` 默认 `inp`。
- 模板变量来自任务字段、几何、资源和参数；缺失的必需变量使渲染失败。
- `other` 不提供程序专用结果解析；结果通过 `%process`、`btproc result` 支持的通用格式或用户 KV 发布。

`%program other` 使用命名模板生成非内建程序的输入文件。模板决定正文和后缀，`prog` 参数决定运行环境配置。

3.13.3.1 模板目录解析顺序

为了兼顾局部覆盖和全局模板库，`other` 模板目录采用分层查找机制。解析顺序如下：

1. `define: TEMPLATE_DIR`
2. 调用方显式提供的模板目录
3. `BANETASK_OTHER_TEMPLATES_PATH`

3.13.3.2 模板文件命名

模板的文件名与参数命名直接决定系统能否正确定位模板并选择环境配置。为保证行为可预期，应遵循以下规则：

- 规范写法是 `%template <name>`；`job=<name>` 是可由迁移工具收敛为 `%template` 的输入别名。
- 模板文件名为 `<job>.<suffix>`。
- `suffix` 默认 `inp`，可由 `%param` 或替代参数中的 `suffix=<ext>` 覆盖。
- 任务参数或模板默认段中应提供 `prog=<program>`，用于选择 `<program>.conf` workflow 环境配置。

3.13.3.3 模板默认参数

模板正文之后可追加 `-default-` 段，用来声明一组默认参数。默认参数会减少任务文件中的重复书写，并使模板本身接近可直接复用的标准片段。

```
! [functional] [basis]
* xyzfile [charge] [spin] [xyz_file]

-default-
prog=orca
nprocs=32
maxcore=3000
functional=wB97M-V
basis=def2-TZVP
```

规则如下：

- `-default-` 之后每行一个 `key=value`。
- 用户未显式提供的参数会自动采用默认值。
- `%control` 的资源设置会在此基础上再次覆写 `nprocs`、`memory`、`maxcore`。

3.13.3.4 模板可用占位符

`other` 模板可直接使用以下占位符：

- `[key]`：来自 `%param`、`%keywords` 替代参数或模板默认段的用户参数。
- `[xyz_file]`：任务目录中自动生成的标准化 XYZ 文件名。
- `[charge]`：源任务电荷；若源任务未提供，则回退到 origin；origin 也未提供时告警并使用 `0`。应用 `%mod` 后为最终值。
- `[spin]`：源任务自旋多重度；若源任务未提供，则回退到 origin；origin 也未提供时告警并使用 `1`。应用 `%mod` 后为最终值。
- `[uhf]`：`[spin] - 1`，适合 XTB `--uhf` 等场景。
- `[geom]`：几何坐标正文。
- `[source]`：主几何的来源描述。

- `[atom_count]`：主几何原子数。
- `[rootname]`、`[taskname]`、`[originname]`：生成阶段的内置占位符。
- `[inputname]`、`[sourcename]`：生成阶段的通用任务占位符。

当 `%source` 声明了 `geom.<slot>` 时，模板还可使用 `[geom.<slot>]`、`[xyz_file.<slot>]`、`[charge.<slot>]`、`[spin.<slot>]`、`[uhf.<slot>]`、`[source.<slot>]` 与 `[atom_count.<slot>]`。完整规则见命名几何来源。这些系统名不能作为 `%param` 键。

3.13.3.5 `other` 生成物

`other` 任务会生成模板输入文件和任务 manifest；模板或运行命令需要 `[xyz_file]` 时还会保留标准化 XYZ。具体包括：

- `<task>_<origin>.<suffix>`
- `<task>_<origin>.xyz`（仅在它是实际输入或被生成产物引用时保留）
- `task.manifest.json`

`task.manifest.json` 的 `program_params` 会记录以下常用键：

- `prog`
- `nprocs`
- `memory`
- `maxcore`

workflow 会优先使用当前任务和模板解析得到的参数；必要时再读取 `task.manifest.json` 中的 `program_params`，决定使用哪个环境配置，以及采用 `memory` 还是 `maxcore` 的资源模式。其中 `prog=<program>` 决定 `<program>.conf`；若缺少该值，`other` 任务虽然仍可以生成模板输入文件，但 workflow 将无法确定应使用哪一个环境配置文件。

3.13.4 运行与结果

- 模板查找依次考虑任务定义、显式模板目录和 `BANETASK_OTHER_TEMPLATES_PATH`。
- 运行命令、输入后缀、输出后缀和结果处理均由所选 `<name>.conf` 与任务 Process DSL 决定。

3.13.5 失败条件与诊断

- 模板不存在、`prog` 缺失、占位符未定义或配置文件不可找到时，准备阶段失败。
- 程序成功但结果为空时，检查模板生成物、所选 `.conf` 和 `%process` 是否显式发布结果。

4 Workflow 运行管理

本章说明配置查找、程序环境、Process DSL、任务准备、`.bwrk`、本地与调度器执行、状态和诊断。

4.1 配置与运行环境

运行时配置决定 BaneTask 在本机或集群环境中的实际行为，包括模板目录、程序环境、默认资源、队列目录和调度 profile。常用配置建议集中放在 `~/.bane/task/`，项目内只保留与该项目强相关的入口文件和少量覆盖项。

标准目录布局：

```
~/.bane/task/
  banetask.conf
  envs/
  bt_templates/
  btlib/
  other_templates/
  scripts/
  multiwfn_templates/
  banewfn/
  external/
  btrun/
    profiles/
    queues/
    routes/
~/.bane/taskq/
  pending/
  running/
  done/
  failed/
  logs/
```

4.1.1 配置文件 `banetask.conf`

`banetask.conf` 保存全局路径和默认运行选项。它使用单行 `KEY=value` 格式，支持 `$HOME` / `${HOME}` 形式的环境变量展开；以 `#` 开头的整行是注释。不要在值后面追加行内注释，因为解析器会把它视为值的一部分。

4.1.1.1 查找顺序与优先级

BaneTask 按下列顺序查找 `banetask.conf`，找到第一个存在的文件后停止：

1. 当前目录
2. `~/.bane/task/`
3. `~/.banetask/`
4. `banetask` 可执行文件所在目录
5. 相对于可执行文件的 `../share/banetask/conf/`
6. `/etc/banetask/`（非 Windows）

同名配置项的优先级为：

环境变量 > 配置文件 > 内置默认值

因此，长期默认值适合写入 `~/.bane/task/banetask.conf`；一次性覆盖适合在 shell 中直接导出环境变量。

用于**查找文件、模板、共享库片段或可执行候选**的路径配置项支持路径列表：POSIX 平台与 `PATH` / `LD_LIBRARY_PATH` 一样使用冒号 `:` 分隔，Windows 使用分号 `;` 分隔以避免和 `C:\` 盘符冲突。BaneTask 会按从左到右的顺序查找，先命中的文件或目录优先；因此可把自定义目录放在随程序安装目录之前，例如：

```
BANETASK_BTLIB_PATH=$HOME/custom/btlib:${BANETASK_CONFIG_DIR}/btlib
```

用于输出或状态存储的单一路径（例如 **BANETASK_LOG_PATH**、**BTRUN_QUEUE_DIR**）仍按单一路径处理。

4.1.1.2 BaneTask 基础路径

- **BANETASK_TASK_PATH**：未显式指定路径运行 **banetask** 时的任务搜索根目录。
- **BANETASK_LOG_PATH**：BaneTask 自身日志输出目录；留空时写在对应 **.bt** 文件所在目录。
- **BANETASK_ENVCONF_PATH**：程序环境配置目录，保存 **g16.conf**、**orca.conf**、**amesp.conf**、**bdf.conf**、**xtb.conf** 等 **<type>.conf** 文件。支持路径列表；**banetask** 生成 workflow、**benv** 命令和 **btrun task** 都会按目录顺序查找。
- **BANETASK_EXTERNAL_SCRIPTS_PATH**:外部程序环境脚本目录。支持路径列表;workflow 初始化时按目录顺序加载第一个存在的 **env.sh**（Bash）或 **env.bat** / **env.cmd**（Cmd），随后才执行程序配置中的 **ENV_SETUP**。该加载不依赖任务是否生成 **pre_comd** 或 **comd**。

4.1.1.3 模板与工具路径

- **BANETASK_TEMPLATE_PATH**：**.bt** 模板目录；供 **btool template** 使用，也作为 **btc** 交互式 **ask:** 菜单查找变量模板的默认目录。支持路径列表。
- **BANETASK_BTLIB_PATH**：**&INCLUDE <...>** 查找 **.btlib** 与共享片段文件的目录，默认 **~/.bane/task/btlib**。支持路径列表。
- **BANETASK_OTHER_TEMPLATES_PATH**：**%program other** 的输入模板目录。支持路径列表。
- **BANETASK_FCCLASSES_INPUT_TEMPLATE**：可选的 FCclasses3 flexible input 模板文件。留空时只写出 **$$$** 和当前作业必需字段，其余参数使用 FCclasses3 默认值。
- **BANETASK_FCCLASSES_GEN_STATE**：**gen_fcc_state** 可执行文件名或路径，默认 **gen_fcc_state**。
- **BANETASK_FCCLASSES_GEN_DIPFILE**：**gen_fcc_dipfile** 可执行文件名或路径，默认 **gen_fcc_dipfile**。
- **BANETASK_FCCLASSES_DEFAULT_COORDS**: FCclasses 工作流默认坐标, 可取 **CARTESIAN** 或 **INTERNAL**。留空时先采用模板中的 **COORDS**，模板也未设置时使用 **CARTESIAN**。
- **BANETASK_SCRIPTS_PATH**：命令 DSL 中 **run** 的脚本目录；**scripts** 是别名。**btc** 查找默认 runner 时也会搜索其中的 **btrun**。支持路径列表。
- **BANETASK_WFN_EXAMPLES_PATH**：Multiwfn 标准输入模板目录。支持路径列表。
- **BANETASK_BANEWFN_PATH**：**banewfn** 模板目录。支持路径列表。
- **BANETASK_BTRUN_PATH**：显式指定 **btrun** 可执行文件路径或候选路径列表；**autorun**、**btc** 默认 runner、**btool project init** 与 **btool project rebuild --run-script** 会优先读取它。
- **BANETASK_BTDB_PATH**: 显式指定 **btdb** 可执行文件路径或候选路径列表; 启用 **finalize** 时，隐式收尾任务会优先读取它，未设置时回退到 **btdb**。
- **BANETASK_BTPROC_PATH**: 显式指定 **btproc** 可执行文件路径或候选路径列表。workflow 状态写入、结果链路、FCclasses 速率报告和 **finalize.method.generate** 优先读取该项。
- **BANETASK_RUNTIME_CLI_PATH**: 共享的运行时 CLI 候选路径列表。当未设置 **BANETASK_BTPROC_PATH** 时，生成的 workflow 与方法学收尾动作用它定位 **btproc**；若仍未设置，

则优先查找 `banetask` 同目录的 `btproc`，最后使用 `PATH` 中的 `btproc`。该项不是目录，列表中的每一项都应是可执行文件候选路径。

- `BANETASK_GNUPLOT`：Plot DSL 使用的 gnuplot 可执行文件，默认 `gnuplot`；命令行 `--gnuplot` 可覆盖。
- `BTMETHOD_REFS_PATH`：`btproc method` 使用的方法参考库 `references.db` 默认路径或候选路径列表。

4.1.1.4 BaneTask 运行选项

- `BANETASK_ALERT_NTFY`：ntfy 主题名；当主命令失败且 `%control alert` 未关闭时发送失败通知。留空表示关闭通知。
- `BANETASK_SHELL`：命令脚本生成模式，可取 `bash`、`sh`、`gitbash`、`cmd`、`bat`、`dos`。
- `BANETASK_CMD_ENABLE_DIRECTIVES`：是否解析 `%preprocess` / `%process` 中的命令 DSL。未设置时默认启用；关闭后，内建命令和自定义插件都不解析，块内每一行均按原样 shell 命令处理。
- `BANETASK_PROCESS_PLUGINS_ENABLED`：是否加载 manifest 型自定义 Process DSL 插件。未设置时默认启用；`false`、`0`、`no`、`off` 等非真值关闭自定义插件，不影响内建命令。
- `BANETASK_PROCESS_PLUGIN_PATH`：自定义 Process DSL 插件的附加搜索根或 manifest 文件列表。POSIX 接受冒号 `:` 或分号 `;` 分隔，Windows 使用分号 `;`；每一项都执行环境变量展开。
- `BANETASK_PROCESS_PLUGIN_ALLOW_OVERRIDE`：是否允许自定义插件覆盖同名内建命令。默认关闭；启用后，自定义 alias 的优先级高于内建 alias。
- `BANETASK_PROCESS_PLUGIN_STRICT`：把插件发现、manifest 解析和 action 编译诊断提升到错误日志级别。该开关不使整个 `banetask` 调用自动中止；不可加载的 manifest、命令或 action 仍会被跳过。
- `USE_GITBASH`：Windows shell 配置别名；配置值会映射到 `BANETASK_SHELL`。

4.1.1.5 变量别名

- `BANETASK_RUNSCRIPT` / `BANETASK_RUN_SCRIPT` / `BANETASK_RUNSH` / `BANETASK_RUN_SH` / `BANETASK_RUNBAT` / `BANETASK_RUN_BAT`：`btc` 的 `run.sh` / `run.bat` runner 变量别名；优先级低于显式 `--script` 和 `BANETASK_BTRUN_PATH`，高于自动目录搜索。
- `BANETASK_ORCA_TEMPLATES_PATH`：`BANETASK_OTHER_TEMPLATES_PATH` 的别名，用于指定 `%program other` 的模板目录。

4.1.2 `btrun` 配置

`btrun` 负责消费 `.bwrk` workflow、提交调度脚本、管理本地 file queue，并支持 `btrun task -t TYPE` 形式的独立任务提交。它的配置分成三类：全局默认值、`<type>.conf` 任务类型配置、execution profile 调度配置。

4.1.2.1 配置入口

常用入口如下：

- `banetask.conf` 中的 `BTRUN_*` 项：保存默认 backend、profile 目录、本地资源、队列目录等。
- shell 环境变量：临时覆盖 `banetask.conf`，适合一次性提交或测试。
- 命令行选项：单次调用的最高优先级，例如 `--backend`、`--profile-dir`、`--profiles`、`--queue`、`--selection-mode`、`--auto-target`。

- **<type>.conf**：**btrun task -t TYPE** 使用的任务类型配置，决定输入后缀、环境初始化和主命令模板。
- **btrun/profiles/*.yaml**、**btrun/queues/*.yaml**、**btrun/routes/*.yaml**：描述本机、集群或远程 target 的调度配置。

4.1.2.2 btrun setup 分区配置

btrun setup 用于分别查看或编辑不同类别的配置文件。直接运行命令会打开交互式菜单；也可以在命令行中指定配置类别和名称，只处理目标文件，不改动其他配置。

```
# 打开分区菜单
btrun setup

# 编辑任务类型配置 ~/.bane/task/envs/orca.conf
btrun setup task orca

# 分别编辑同名 execution profile、queue catalog 和 routing policy
btrun setup profile wzhc
btrun setup queue wzhc
btrun setup route wzhc

# 编辑本地 target 配置 profiles/local.yaml
btrun setup local
```

可选分区如下：

分区	目标文件	用途
global	**banetask.conf**	全局默认值和配置搜索路径
task NAME	**envs/NAME.conf**	独立任务的环境初始化与运行命令模板
scheduler NAME	**schedulers/NAME.yaml**	调度器命令、脚本后缀和状态映射；**NAME** 可取 **local**、**slurm**、**sge** 或 **pbs**
profile NAME	**profiles/NAME.yaml**	execution target、默认队列和远程连接设置
queue NAME	**queues/NAME.yaml**	队列资源、限制和原生调度头
route NAME	**routes/NAME.yaml**	按资源请求选择队列的规则
local	**profiles/local.yaml**	本地 runner 的启用状态、资源和保留量
paths	不编辑文件	显示当前解析出的配置路径
init	由初始化向导决定	运行调度器探测和整套配置生成向导

目标文件不存在时，**setup** 默认创建一个可编辑的起始模板。任务模板中的运行命令会以退出码 2 中止，必须替换为实际程序命令后才能提交任务。使用 **--no-create** 可要求文件必须已经存在；**--print-path** 只打印目标路径，不创建或打开文件；**--list** 列出分区或该分区中已有的配置文件。

编辑器按以下顺序确定：`--editor COMMAND`、`VISUAL`、`EDITOR`，最后在 POSIX 系统使用 `vi`，在 Windows 使用 `notepad`。需要等待图形编辑器关闭时，应把等待参数包含在编辑器命令中，例如：

```
btrun setup queue wzhc --editor "code --wait"
```

默认路径可分别覆盖：

- `-O DIR` 或 `--config-root DIR`：设置包含 `schedulers`、`profiles`、`queues` 和 `routes` 的配置根目录。
- `--task-config-dir DIR`：设置 `<type>.conf` 所在目录。
- `--global-config FILE`：指定要编辑的 `banetask.conf`。

`btrun setup init [OPTIONS]` 会把后续参数交给 `btrun init`。`init` 适合首次探测调度器并生成一组关联配置；`setup` 的其他分区适合随后独立维护其中一个文件。

4.1.2.3 常用环境变量

- `BTRUN_PROFILE_DIR`：execution profile 目录。支持路径列表；profile 可描述 local runner、本机调度 target 或带 `remote:` 的 SSH 调度 target，并关联 queue catalog、routing policy、提交命令和状态映射。
- `BTRUN_DEFAULT_BACKEND`：target 选择时的全局 backend 偏好，可取 `auto`、`local`、`slurm`、`sge`、`pbs`。`auto` 会发现本机 local runner、本机调度器和已配置的 remote profile，再按任务资源选择 target 与 queue。
- `BTRUN_ALLOW_AUTO_REMOTE`：是否允许未显式指定 profile/backend 的提交自动选择 remote profile。默认关闭；单次调用可用 `--auto-target` 开启，或用 `--no-auto-target` 覆盖并关闭。
- `BTRUN_SELECTION_MODE`：target + queue 的全局选择策略，可取 `local_first`、`remote_first`、`best_fit`、`first_runnable_by_priority`；CLI 的 `--selection-mode` 优先。
- `BTRUN_TASK_CONFIG_DIR`：`btrun task` 查找 `<type>.conf` 的单次覆盖目录。支持路径列表。
- `BTRUN_QUEUE_DIR`：`btrun queue run/status/cancel/kill` 的本地队列目录；为空时读取别名 `BANETASKQ_DIR`，再回退到 `~/.bane/taskq`。
- `BTRUN_LOCAL_QUEUE_DIR`：仅在 scheduler allocation local context 中覆写嵌套 local 队列目录。通常不必设置；默认目录位于系统临时目录，并按用户、调度作业、array 元素和计算节点隔离。
- `BTRUN_TOTAL_CORES` / `BANETASK_TOTAL_CORES`：local runner 可用总核心数；未设置时回退到硬件并发数。local profile 中的 `local.max_cores` 或别名 `defaults.total_cores` 可覆盖，`local.reserve_cores` 会从可调度容量中扣除。
- `BTRUN_TOTAL_MEMORY_MB` / `BANETASK_TOTAL_MEMORY_MB` / `BTRUN_TOTAL_MEM_MB` / `BANETASK_TOTAL_MEM_MB`：local target 的总内存上限。local profile 中的 `local.max_memory_mb` 可覆盖，`local.reserve_memory_mb` 会从可调度容量中扣除。
- `BTRUN_TOTAL_GPUS` / `BANETASK_TOTAL_GPUS`：local runner 可用 GPU 总数；未设置时按 0 处理。local profile 中的 `local.max_gpus` 或别名 `defaults.total_gpus` 可覆盖，`local.reserve_gpus` 会从可调度容量中扣除。
- `BTRUN_GPU_IDS` / `BANETASK_GPU_IDS`：local runner 可分配 GPU ID 列表，例如 `0,1`。local profile 中的 `local.gpu_ids` 或别名 `defaults.gpu_ids` 可覆盖。
- `BTRUN_LOCAL_CONTEXT` / `BANETASK_LOCAL_CONTEXT`：把当前进程标记为 scheduler al-

location 内的嵌套 local context。`btrun` 生成的 Slurm / SGE / PBS 脚本会自动设置该标记，并把本次 allocation 的每节点核心数、内存、GPU 数和可见 GPU 写入上述 `BTRUN_TOTAL_*` / `BTRUN_GPU_IDS` 变量；一般不应在登录节点手动设置。

- `BTRUN_FOLLOW_LOGS` / `BANETASK_FOLLOW_LOGS`：local backend / local queue 执行时是否跟随 `.out` / `.err` 日志。
- `BTRUN_HIGH_MEMORY_THRESHOLD_MB`：`btrun task` 处理高人均内存任务时使用的阈值。
- `BTRUN_DEFAULT_CORES`：`btrun task` 在 CLI、manifest 和 `<type>.conf` 都没有给出核心数时使用的默认核心数。
- `BTRUN_DEFAULT_CORES_FOR_MAX_MEM`：`btrun task` 遇到高 `maxcore` / 高人均内存任务时，用于估算 cluster `cores` 的默认值。
- `BTRUN_PHYSICAL_CPU_NUM`：把 `nprocs` / `memory` / `maxcore` 映射为节点数与每节点核心数时使用的物理 CPU 参考值。
- `BTRUN_INIT_SKIP_GPU_PROBE`：真值时，`btrun init` 不调用 `nvidia-smi` 探测 GPU；仍可用 `--local-gpus` 和 `--local-gpu-ids` 显式写入资源。
- `BTRUN_SKIP_GPU_PROBE`：`BTRUN_INIT_SKIP_GPU_PROBE` 的配置别名。任一变量为真值都会跳过 GPU 探测。
- `BTRUN_DEBUG_BUNDLE_ON_FAIL` / `BANETASK_DEBUG_BUNDLE_ON_FAIL`：local queued job 失败时是否收集调试 bundle。
- `BTRUN_DEBUG_BUNDLE_DIR` / `BANETASK_DEBUG_BUNDLE_DIR`：调试 bundle 输出目录；未设置时使用任务目录下的默认位置。

4.1.2.4 `btrun task` 的 `<type>.conf`

`btrun task -t TYPE` 会读取 `TYPE.conf` 来决定扫描什么输入文件、如何加载环境、运行什么命令。查找顺序如下：

1. `--config-dir DIR`
2. `BTRUN_TASK_CONFIG_DIR`（路径列表按从左到右展开）
3. `BANETASK_ENVCONF_PATH`（路径列表按从左到右展开）
4. `./envs`
5. `./conf/envs`
6. 安装目录中的 envs
7. `~/.bane/task/envs`

`TYPE.conf` 不是所有内容都写成 `KEY=value` 的平面配置。文件包含两类彼此并列的节：

- `[main]` 只放标量配置，每项写成 `KEY=value`。
- `[ENV_SETUP]`、`[RUN_CMD_TEMPLATE]` 和自定义 `[NAME]` 是位于 `[main]` 之外的原样脚本区块。区块正文直接写命令，不写成赋值语句。

4.1.2.4.1 `[main]` 标量字段

下列项目写在 `[main]` 中：

参数名	说明
`DESCRIPTION`	`btrun task --help-type TYPE` 显示的说明文本
`SUFFIX`	按后缀扫描输入；与 `INPUT_FILENAME` 互斥，task mode 中两者均未配置时默认 `inp`

参数名	说明
INPUT_FILENAME	按 basename 精确扫描固定文件名输入；与 SUFFIX 互斥
OUTPUT_SUFFIX	主输出文件后缀，默认 log
CONF_CORES	默认单任务核心数
CONF_MEMORY	默认单任务内存，单位 MB
CONF_SPAN_NODES	程序是否允许跨节点；task mode 默认 false
RAW_MODE	true / false；为 true 时每个输入目录作为一个 raw task 提交
ARRAY	后端支持 array job 时使用的数组规模
USE_ABSOLUTE_PATH	是否在命令模板中使用绝对输入/输出路径

固定文件名示例可写为 INPUT_FILENAME=MINP 或 INPUT_FILENAME=comd。该值必须是单个 basename，不接受目录分隔符；其中的字符按字面量精确匹配，不做通配符展开。配置了它以后必须删除 SUFFIX。

4.1.2.4.2 [ENV_SETUP]

[ENV_SETUP] 与 [main] 同级，正文是主命令前执行的环境初始化脚本。正文直接写命令，不使用 KEY=value；不需要初始化时写 :。

```
[ENV_SETUP]
source "$HOME/apprepo/gaussian/16/env.sh"
```

4.1.2.4.3 [RUN_CMD_TEMPLATE]

[RUN_CMD_TEMPLATE] 与 [main] 同级，正文是主命令模板。它不是 [main] 中的参数，因此不能写成 RUN_CMD_TEMPLATE=...。

```
[RUN_CMD_TEMPLATE]
g16 "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}"
```

正文支持 ${NAME} 与 $NAME 两种占位符；规范形式为 ${NAME}。常用变量包括：

- ${ACTUAL_INPUT_FILE}：解析后的输入路径或文件名。
- ${ACTUAL_OUTPUT_FILE}：解析后的输出路径或文件名。
- ${ACTUAL_INPUT_FILENAME} / ${ACTUAL_OUTPUT_FILENAME}：输入/输出文件名。
- ${ACTUAL_INPUT_STEM}：输入文件去后缀后的 stem。
- ${FILE_NAME} / ${BASENAME}：作业名或文件 stem。
- ${WORKDIR}：作业工作目录。
- ${CORES}：本次任务的有效核心数。
- ${EXTRA_ARGS} / ${ARGS}：--extra-args 或 manifest 中的程序参数。
- ${TYPE}：解析后的任务类型。

4.1.2.4.4 自定义 [NAME]

自定义 [NAME] 也是与 [main] 同级的原样脚本区块，例如 [POST]。btrun task --post-var NAME 选择对应正文作为后处理脚本；正文可使用与 [RUN_CMD_TEMPLATE] 相同的变量。

```
[POST]
banedata "${ACTUAL_OUTPUT_FILE}" -x
```

完整的最小配置如下：

```
[main]
DESCRIPTION=Gaussian via g16
SUFFIX=gjf
OUTPUT_SUFFIX=log
CONF_CORES=32
CONF_MEMORY=96000

[ENV_SETUP]
source "$HOME/apprepo/gaussian/16/env.sh"

[RUN_CMD_TEMPLATE]
g16 "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}"
```

4.1.2.5 execution profile 查找

execution profile 目录查找顺序如下：

- 命令行 `--profile-dir`
- `BTRUN_PROFILE_DIR`
- `./btrun/profiles`
- 可执行文件附近的 `btrun/profiles`
- `~/.bane/task/btrun/profiles`

profile 使用 YAML 描述一个 execution target，而不是把一次提交提前锁死到某个全局 backend。`scheduler`（也接受 `backend` / `kind`）声明 target 的调度类型；`queue_catalog` 与 `routing_policy` 关联该 target 的队列能力和资源路由；`selection.priority` 控制跨 target 排序；`remote:` 块把它标记为 SSH/rsync 远程 target。

调度 profile 的常用字段包括：

- `kind` / `name` / `scheduler`
- `scheduler_config` / `queue_catalog` / `routing_policy`
- `script_extension`
- `submit_cmd` / `query_cmd` / `cancel_cmd` / `discover_queue_cmd`
- `header_lines`
- `state_map`
- `selection.priority`
- `defaults.queue` / `defaults.physical_cpu_num` / `defaults.parallel_env`
- `remote.host` / `remote.root` / `remote.health_check`

4.1.2.6 Slurm profile 示例

典型 Slurm profile 会声明 `scheduler: slurm`，并定义或引用 `sbatch`、`squeue`、`scancel` 以及 `#SBATCH` 头部模板。`btrun queue run/status/cancel/kill` 则读取 `BTRUN_QUEUE_DIR` 管理本地队列。

```yaml
kind: slurm
name: default
script_extension: .slurm
submit_cmd: [sbatch, "{script}"]
query_cmd: [squeue, -h, -j, "{jobid}", -o, "%T"]
cancel_cmd: [scancel, "{jobid}"]
discover_queue_cmd: [sinfo, -h, -o, "%P"]
defaults:
  queue: ""
  physical_cpu_num: 64
header_lines:
  - "#!/bin/bash"
  - "#SBATCH -J {{job_name}}"
  - "#SBATCH --ntasks-per-node={{cores}}"
  - "#SBATCH -N {{nodes}}"
  - "#SBATCH --mem={{memory_mb}}M"
  - "#SBATCH -p {{queue}}"
  - "#SBATCH -o {{workdir}}/slurm_{{job_name}}.out"
  - "#SBATCH -e {{workdir}}/slurm_{{job_name}}.err"
  - "{{array_directive}}"
  - "export CORES={{cores}}"
  - "cd {{workdir_quoted}} || exit 97"
state_map:
  PENDING: queued
  CONFIGURING: queued
  RUNNING: running
  COMPLETED: succeeded
  FAILED: failed
  CANCELLED: cancelled
  TIMEOUT: failed
  NODE_FAIL: failed
```

4.1.2.7 remote profile

调度 profile 可以加入 **remote** 块，使本地 **btrun** 通过 SSH/rsync 把完整 case 暂存到集群、在远端提交脚本，并在本地队列中保存持续跟踪记录：

```yaml
name: remote-slurm
scheduler: slurm
queue_catalog: ../queues/remote-slurm.yaml
routing_policy: ../routes/remote-slurm.yaml
selection:
  priority: 20
remote:
  host: login.cluster.example
  user: username
  port: 22
  root: /scratch/username/banetask_remote
  results_dir: Results
  health_check: false
```

```
  ssh_cmd: [ssh]
  rsync_cmd: [rsync, -az]
  stage_excludes: [Results]
```

`selection.priority` 是跨 target 排序时使用的 profile 优先级，数值越小越优先；queue 自己的 `priority` 仍用于 profile 内和跨 target 的候选比较。remote profile 的发现不依赖本机是否安装 `squeue`、`sbatch` 或 `qsub`，默认发现阶段也不会连接 SSH；只有 target 被选中并提交，或启用了健康检查时，才访问远端。`remote.health_check: true` 等价于默认启用提交前 SSH scheduler 检查，CLI 的 `--health-check` / `--no-health-check` 可对单次调用覆盖。

远程记录的完成/失败判定以远端 case 的 `.bane/run/chain.status.json` 为主状态源，并用 chain 的 `running` 摘要所引用的 `.bane/run/tasks/*.json` 校正刚刚发生、尚未回写到 chain 摘要中的任务终态。本次提交任务自己的 task status 还用于确认任务确实启动，并在 chain 尚未写出时捕获明确失败。这个状态机不依赖已经从 `squeue`、`qstat` 等活动队列表中消失的作业；`query_cmd` 只用于显式的 `btrun status <backend> <jobid>` 查询。

每次新的本地提交进程会在 `remote.root/<case-key>/<session-id>/` 下建立独立 session；同一次 `submit` / `kick` 中属于同一 case 的初始任务共享这个目录。远程记录还会保存暂存时的 chain/task 指纹，只有本次提交的 task 状态确实推进、且 chain 状态也推进后，才接受成功终态，从而避免把本地或远端遗留的旧 `.bane/run`、旧 `Results` 误认为本次运行结果。若远端一直没有生成或推进 banetask chain 状态，成功任务的记录会继续保持 queued/running；明确的 task failure 仍会进入失败终态。

`queue run` 会在远程任务完成后把远端 `Results/`、`.bane/store/`、`.bane/cache/snapshots/` 与 `.bane/run/` 拉回本地 case，并把记录移入 `remote/done` 或 `remote/failed`。其中 `.bane/cache/project.db` 不会从远端回收；需要查询时由本地 records、revisions 与对象存储重新同步或重建。这份 case 级记录跟踪整条 banetask chain，而不是只跟踪第一次 scheduler job。

4.1.2.8 local profile

local target 的推荐配置放在 `btrun/profiles/local.yaml`。`btrun init` 会自动生成它，用于声明这台机器是否应该参与 local backend 调度，以及 local runner 的可用资源和预留资源：

```
kind: btrun_profile
schema_version: 1

name: local
scheduler: local
script_extension: .btq

selection:
  priority: 20

defaults:
  queue: local
  default_cores: 16
  default_memory_mb: 50000

local:
```

```
  enabled: true
  role: "server"      # login_node / workstation / server / small_host / auto
  max_cores: 64
  max_memory_mb: 262144
  max_gpus: 2
  gpu_ids: "0,1"
  reserve_cores: 4
  reserve_memory_mb: 16384
  reserve_gpus: 0
```

```
btrun submit -L ./case/opt.bwrk
btrun queue run --poll 5
```

local target 使用 `.btq` 作为可检查的 workflow 脚本后缀。提交已有 `.bwrk` 时，脚本写入该 job 的工作目录并沿用原文件名 stem；`btrun task` 按输入文件或 raw task 名称生成对应的 `.btq`。命令输出中的 `script=` 指向这份文件。`--dry-run` 也会写出 `.btq`，但不会创建待运行 job，也不会删除作为输入的 `.bwrk`。

本地队列同时保存一份独立的 `.bwrk` 运行副本。runner 只消费队列副本，因此提交后查看、移动或修改工作目录中的 `.btq` 不会改变已经排队的作业。`script_extension` 可指定其他留档后缀；未配置时使用 `.btq`。

`local.enabled: false` 会让 local target 在 target-first 选择中被拒绝，并在 `--explain` 中显示原因；这适合 2 核/2G 等集群登录节点。`max_cores` / `max_memory_mb` / `max_gpus` 是本机总资源，`reserve_cores` / `reserve_memory_mb` / `reserve_gpus` 会从可调度容量中扣除。`gpu_ids` 控制可分配 GPU 槽位；`high_memory_threshold_mb_per_core` 仍可放在 `defaults:` 下，用于把高内存 `.bwrk` 或 task 动态折算为更多核心。`default_cores` 与 `default_memory_mb` 仅在 `btrun task` 没有从命令行、manifest 或 `<type>.conf` 读取到资源默认值时生效。`btrun/profiles/local/<name>.yaml` 与 `defaults.total_cores` / `defaults.total_gpus` / `defaults.gpu_ids` 也可读取；新建配置建议使用顶层 `profiles/local.yaml` 加 `local:` 块。

在 scheduler allocation local context 中，上述登录节点 profile 的 `local.enabled`、`max_*` 与 `reserve_*` 不再代表当前 local 容量；`btrun` 改用调度脚本导出的本次 allocation 资源，并使用独立的临时 file queue。这个 allocation-local target 对普通任务的优先级低于正常调度 target；`.bwrk` 带 `localfirst: true` 时，只有在资源可立即取得的情况下才会抢先。若 `CUDA_VISIBLE_DEVICES` 等变量使用 GPU UUID、MIG 标识或其他无法安全拆分为数字槽位的值，嵌套 local GPU 调度会保守关闭并回退到通常的 scheduler target，CPU-only 任务不受影响。

4.1.3 程序环境配置文件

BaneTask 生成 workflow 时会读取 `BANETASK_ENVCONF_PATH` 下的程序环境配置。每种程序通过配置文件决定输入后缀、环境初始化方式和主命令模板；任务文件描述计算流程，程序调用细节由这些配置文件提供。

4.1.3.1 文件格式

程序环境配置文件不是平面的 `KEY=value` 列表，而是由两类并列的节组成：标量配置集中写在 `[main]`；shell、批处理或其他命令文本写在 `[main]` 之外的独立原样区块中：

```
[main]
CONF_CORES=32
CONF_MEMORY=96000
SUFFIX=gjf

[ENV_SETUP]
source "$HOME/apprepo/gaussian/16/env.sh"
export PATH="$HOME/scripts/bin:$PATH"

[RUN_CMD_TEMPLATE]
g16 "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}"
```

解析规则如下：

- **[main]** 中每个有效配置项写成 **KEY=value**。空行和首个非空字符为 **#** 的整行被忽略；行内 **#** 属于值本身。
- **[ENV_SETUP]**、**[RUN_CMD_TEMPLATE]** 以及其他 **[NAME]** 都是与 **[main]** 并列的区块标题，不是 **[main]** 中的参数。其正文按原文保存，单引号、双引号、**#**、**=**、**;** 和换行均不承担配置语法。**awk '...'**、here document 和多条命令可直接书写，无需再包裹外层引号。
- 区块标题必须从行首开始并独占一行；名称由字母、数字和下划线组成，且首字符不能是数字。新的区块标题结束前一个原样区块。
- 原样区块开头和结尾仅用于排版的空白行不计入正文；区块内部的空行和缩进会保留。
- 自定义 **[NAME]** 的标题名可供 **btrun task --post-var NAME** 选择对应区块。

BaneTask 生成 managed workflow 与 **btrun task** 共用同一类 **<type>.conf**。下面分别说明 **[main]** 内的标量字段和 **[main]** 之外的原样脚本区块。**DESCRIPTION**、**RAW_MODE**、**ARRAY**、**USE_ABSOLUTE_PATH** 主要服务于 **btrun task**；资源字段也会参与 target/queue 选择。

4.1.3.1.1 [main] 标量字段

下列字段必须写在 **[main]** 中，格式为 **KEY=value**：

参数名	managed workflow 必需?	说明	默认值
DESCRIPTION	n	**btrun task --help-type TYPE** 显示的类型说明	—
SUFFIX	二选一	按输入后缀检索，例如 **gjf**、**inp**、**sh**；与 **INPUT_FILENAME** 互斥	**inp**（仅 **btrun task**）
INPUT_FILENAME	二选一	按 basename 精确检索固定文件名，例如 **MINP** 或 **comd**；与 **SUFFIX** 互斥	—
OUTPUT_SUFFIX	n	主输出文件后缀	**log**
CONF_CORES	n	默认单任务核心数	—
CONF_MEMORY	n	默认单任务总内存，单位 MB	—

参数名	managed workflow 必需?	说明	默认值
`CONF_SPAN_NODES`	n	程序是否允许跨节点；只有程序确实支持多节点并行时才设为 `true`	`false`（task mode）
`RAW_MODE`	n	`true` 时把每个输入目录作为一个 raw task，而不是扫描普通输入文件	`false`
`ARRAY`	n	backend 支持 scheduler array 时使用的数组规模	`0`
`USE_ABSOLUTE_PATH`	n	命令模板中的输入/输出变量是否展开为绝对路径；`false` 时仅使用文件名	`true`

managed workflow 必须显式配置 `SUFFIX` 或 `INPUT_FILENAME`，且只能配置其中一个。独立 `btrun task` 为兼容旧配置，在两者均缺失时仍按 `SUFFIX=inp` 扫描。`INPUT_FILENAME` 只接受单个 basename，不接受目录路径，也不做通配符展开。

`CONF_SPAN_NODES` 还兼容 `CONF_ALLOW_NODE_SPAN`、`CONF_MULTI_NODE`、`CONF_MULTINODE` 与 `CONF_NODE_SPAN` 等旧名称；新配置统一使用 `CONF_SPAN_NODES`。

4.1.3.1.2 `[ENV_SETUP]`

`[ENV_SETUP]` 与 `[main]` 同级，是 managed workflow 必需的环境初始化区块。正文直接写 shell、批处理或相应平台命令；无需初始化时写 `:`。单引号、双引号、`#`、`=`、`;` 和换行都按原文保留。

```
[ENV_SETUP]
source "$HOME/apprepo/gaussian/16/env.sh"
export PATH="$HOME/scripts/bin:$PATH"
```

4.1.3.1.3 `[RUN_CMD_TEMPLATE]`

`[RUN_CMD_TEMPLATE]` 与 `[main]` 同级，是 managed workflow 必需的主命令区块。正文支持 `${NAME}`，未加花括号的 `$NAME` 也可解析；规范形式为 `${NAME}`。可用变量见Workflow 中的 RUN_CMD_TEMPLATE 可用变量。

```
[RUN_CMD_TEMPLATE]
g16 "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}"
```

`ENV_SETUP=...` 和 `RUN_CMD_TEMPLATE=...` 不应写在 `[main]` 中。

4.1.3.1.4 自定义 `[NAME]`

自定义 `[NAME]` 与 `[main]` 同级，用于保存可选脚本，例如供 `btrun task --post-var NAME` 选择的后处理区块。区块标题必须从行首开始并独占一行，名称由字母、数字和下划线组成，首字符不能是数字。

4.1.3.2 Gaussian 示例

```
[main]
# 默认 32 核，96GB
```

```
CONF_CORES=32
CONF_MEMORY=96000
# 输入文件后缀 gjf
SUFFIX=gjf

[ENV_SETUP]
source "$HOME/apprepo/gaussian/16-hy/scripts/env.sh"

[RUN_CMD_TEMPLATE]
# 使用 g16loop 包装脚本运行 g16，并在运行结束后用 banedata -x 提取末次优化结构
g16loop "${ACTUAL_INPUT_FILE}" --crash-handle all
banedata "${ACTUAL_OUTPUT_FILE}" -x
```

4.1.3.3 环境配置文件选择规则

banetask 内置程序类型对应的环境配置文件选择逻辑如下：

- **gaussian**：依次尝试 **g16.conf**、**gaussian.conf**。
- **orca**：**orca.conf**。
- **gamess**：依次尝试 **gamess.conf**、**gms.conf**。
- **amesp**：**amesp.conf**。
- **bdf**：**bdf.conf**。
- **mrcc**：**mrcc.conf**，内置示例使用 **INPUT_FILENAME=MINP**。
- **maple**：**maple.conf**。
- **xtb**：**xtb.conf**。
- **script**：**script.conf**。
- **other**：优先读取 **task.manifest.json** 中的 **program_params.prog**，再选择 **<prog>.conf**。

4.1.4 模板、片段库与脚本目录

模板和脚本目录保存可复用的任务模板、输入模板、片段库和后处理脚本。它们通常位于 **~/.bane/task/** 下，由 **banetask.conf** 中的路径项指定。

4.1.4.1 .bt 模板与变量模板

BANETASK_TEMPLATE_PATH 供 **btool template ...** 子命令使用，也作为 **btc** 交互式 **ask:** 菜单查找变量模板的默认目录。模板管理器会递归扫描目录中的 **.bt** 文件，并按以下规则组织：

- 文件名去后缀后作为模板名。
- 所在子目录名作为模板分类；若模板直接位于根目录，则分类记为 **General**。
- **btool template merge** 会按模板中的任务块做合并；当同名块的 **%source** 与 **%keywords** 一致时，会自动尝试并块，并对 **%process** / **%preprocess** 序列做顺序保持的合并。
- **btool template variable** 会扫描模板中的 **[name]** 占位符，排除内置保留名后生成 **define:** 变量草案。

同一目录下的 **.yaml** / **.yml** 文件可作为 **btc** 的变量模板被递归扫描。格式如下：

```
name: B3LYP/def2-SVP
description: 快速载入常用方法组
variables:
  func: b3lyp
  basis: def2-svp
  scrf: sdd
```

其中 `variables:` 里只有与当前 `.btc` 模板 `ask:` 块同名的键才会被应用。

4.1.4.2 `.btlib` 片段库

`BANETASK_BTLIB_PATH` 保存可被 `&INCLUDE <...>` 查找的共享片段库。默认位置是 `~/.bane/task/btlib`。适合放入跨项目复用的控制块、后处理命令、任务体片段或完整任务片段，例如：

```
~/.bane/task/btlib/
  analysis.btlib
  normal_controls.btlib
  workflows/
    td.btlib
```

任务文件中使用尖括号形式引用共享库：

```
&INCLUDE <analysis.btlib> as analysis

$td
  %process
    &INCLUDE analysis.hole_ele(dir="excit")
```

相对路径形式 `&INCLUDE "lib/analysis.btlib" as analysis` 始终相对于写出该 include 的源文件解析；尖括号形式则从 `BANETASK_BTLIB_PATH` 的路径列表按从左到右查找，未设置时回退到默认 `~/.bane/task/btlib`。

4.1.4.3 `other` 输入模板

`BANETASK_OTHER_TEMPLATES_PATH` 供 `%program other` 使用。模板解析顺序如下：

1. `define: TEMPLATE_DIR`
2. 调用方显式提供的模板目录
3. `BANETASK_OTHER_TEMPLATES_PATH`

上述每一层模板目录都可以是路径列表；同名模板在靠前目录中存在时会优先生效。

模板文件名采用 `<template>.<suffix>` 形式，其中 `<template>` 来自 `%template`，`<suffix>` 默认 `inp`，可由 `%param suffix=<ext>` 覆盖。模板正文可包含 `-default-` 段声明默认参数，供 `%param` 与 `%control` 在生成阶段继续覆写。

4.1.4.4 命令脚本目录

`BANETASK_SCRIPTS_PATH` 指定命令 DSL 的项目脚本目录。在 `%process` / `%preprocess` 中使用 `run post1` 时，解析器会按路径列表顺序查找 `post1.sh` 或 `post1.bat` 并调用第一个命中的脚本。`scripts post1` 是 `run post1` 的别名。

4.1.4.5 Multiwfn 与 banewfn 模板

`BANETASK_WFN_EXAMPLES_PATH` 与 `BANETASK_BANEWFN_PATH` 分别服务于命令 DSL 中的

multiwfn / mwfn 与 banewfn:

- **BANETASK_WFN_EXAMPLES_PATH** 下放置 **<template>.txt**，命令 DSL 会按路径列表顺序复制第一个命中的文件并将其作为 Multiwfn 的标准输入。
- **BANETASK_BANEWFN_PATH** 下放置 **<base>.bwc** 或 **<base>.inp**。**banewfn** DSL 会优先在当前目录查找模板，其次按 **BANETASK_BANEWFN_PATH** 的路径列表回退查找，并自动补上核心数参数。

这两类目录属于后处理模板库，与主输入生成目录彼此独立。

4.2 Process DSL 与前后处理

在 **%preprocess** 与 **%process** 中，命令 DSL 用统一写法描述任务生成阶段之外的辅助动作，例如复制文件、归档结果、调用模板脚本、切换环境以及提取自定义结果。它为常见操作提供更稳定、跨平台差异更小的写法，同时保留原样 shell 命令入口。

4.2.1 命令解析模型

系统会按顺序逐行解释命令。先尝试把每一行识别为 DSL 指令；识别失败时，该行作为普通 shell 命令原样写入。整体处理顺序如下:

1. 若 **BANETASK_CMD_ENABLE_DIRECTIVES** 为真，则尝试识别 DSL 指令。
2. 未识别为 DSL 指令的行会原样作为 shell 命令写入。
3. 对生成结果执行默认占位符替换。

原始 **%preprocess** / **%process** 块支持续行：行尾单个反斜杠 **** 会把下一行拼接到同一条命令中；双反斜杠或非行尾反斜杠不触发续行。续行中的空白行会被跳过，若文件在续行状态结束，解析器会保留已拼接的命令并给出警告。

默认占位符如下:

- **[rootname]**：当前 case 根名。
- **[taskname]**：当前任务块名。
- **[inputname]**：**<taskname>_<originname>**。
- **[originname]**：原始输入基名。
- **[sourcename]**：**%source** 的当前值。

4.2.2 shell 类型选择

由于 BaneTask 同时支持 POSIX 与 Windows 环境，因此命令脚本和 workflow 的 shell 风格需要根据平台和配置自动选择。判定规则如下:

- **BANETASK_SHELL=bash|sh|gitbash**：使用 Bash 风格脚本。
- **BANETASK_SHELL=cmd|bat|dos**：使用 Cmd 风格脚本。
- Windows 下使用 **BANETASK_SHELL=gitbash**、**bash** 或 **cmd** 选择 shell；**USE_GITBASH=true** 是别名。
- 非 Windows 平台默认使用 Bash。

4.2.3 DSL 指令

4.2.3.1 run

```
run submit
```

`scripts post_analyze arg1 arg2` 是 `run post_analyze arg1 arg2` 的别名。

行为如下：

- `BANETASK_SCRIPTS_PATH` 为单一路径时直接调用对应脚本：Bash 展开为 `bash "$BANETASK_SCRIPTS_PATH/<name>.sh" ...`，Cmd 展开为 `call "%BANETASK_SCRIPTS_PATH%\<name>.bat" ...`。
- `BANETASK_SCRIPTS_PATH` 为路径列表时，生成脚本会按配置顺序查找 `<name>.sh` 或 `<name>.bat`，调用第一个存在的脚本；未找到则报错退出。

4.2.3.2 mwfn / multiwfn

```
mwfn mol.fchk density
multiwfn td.wfn hole-electron -silent
```

行为如下：

1. 从 `BANETASK_WFN_EXAMPLES_PATH` 复制 `<template>.txt` 到当前目录；若该配置是路径列表，则按配置顺序使用第一个存在的模板文件。
2. 调用 `Multiwfn <input>`。
3. 将模板作为标准输入重定向给 Multiwfn。
4. 将输出写入 `mw_<template>_out.txt`。

4.2.3.3 publish

```
publish summary.txt plot/
publish --pin final.svg summary.csv
```

行为如下：

- 目标目录固定为 `../Results/[inputname]`。
- 先创建目标目录，再递归复制给定源路径。
- `--pin` 在复制完成后，从本次源路径匹配结果推导精确目标文件并原子登记到 `Results/published.json`；目标目录中本次没有复制的历史文件不会因通配符同名而被登记。
- 目录源按其中实际复制的普通文件登记；空目录不能 pin。异常退出遗留的清单更新锁可自动识别并回收。
- `--pin` 不根据程序、任务类型或扩展名补充任何隐式文件。缺少 `.fchk`、`.gbw`、`.hess` 等文件本身不是错误。
- `archive ...` 是不带 `--pin` 的 `publish ...` 的别名；它不会创建 `.btdb`。

4.2.3.4 cp

```
cp file1.txt file2.txt backup/
cp -r plot/ figures/
```

行为如下：

- 支持多个源和一个目标。
- 支持 `-r`、`-R`、`--recursive`。
- Bash 下使用 `cp`；Cmd 下自动切换为 PowerShell `Copy-Item`。

4.2.3.5 `mv`

```
mv *.png plot/
```

行为如下：

- 支持多个源和一个目标。
- Bash 下使用 `mv`；Cmd 下自动切换为 PowerShell `Move-Item`。

4.2.3.6 `mkdir` / `mkdirp`

```
mkdir plot tables
mkdirp Results/cache
```

行为如下：

- 两者都表示递归创建目录（`mkdir -p`）。
- Bash 下使用 `mkdir -p`；Cmd 下自动切换为 PowerShell `New-Item -ItemType Directory -Force`。

4.2.3.7 `snapshot` / `mass`

`snapshot` 与 `mass` 用于收集某段后处理命令新产生的一层文件和目录。`snapshot` 记录当前任务目录的快照；`mass(dir)` 将当前目录与最近一次快照比较，把新增的一层文件和目录移动到 `dir`。

```
%process
  snapshot exclude=old.tmp exclude=keep_dir
  run post_analyze
  mass(analysis_result) exclude=run.log exclude=manual_keep.txt
```

`snapshot` 语法如下：

```
snapshot [exclude=<name> ...]
snapshot [--exclude <name> ...]
snapshot [--exclude=<name> ...]
```

行为如下：

- 记录当前目录第一层文件和目录名，不递归扫描子目录内容。
- `exclude` 指定的名称不会写入快照。
- `exclude` 可以重复出现，用于排除多个文件或目录。
- 快照文件与快照排除缓存由 BaneTask 写在当前任务目录内部，文件名分别为 `.bane.process.snapshot` 与 `.bane.process.snapshot.exclude`。

`mass` 语法如下：

```
mass(<dir>) [exclude=<name> ...]
mass <dir> [exclude=<name> ...]
```

行为如下：

- 若缺少最近一次 **snapshot** 生成的快照文件，脚本会报错并停止。
- 自动创建目标目录 **dir**。
- 将当前目录第一层中“不在快照内”的文件和目录移动到 **dir**。
- 自动排除 **.bane.process.snapshot**、**.bane.process.snapshot.exclude** 和目标目录本身。
- **mass** 自身的 **exclude** 项不会被移动。
- 最近一次 **snapshot** 的 **exclude** 项也会在 **mass** 阶段继续排除,因此 **snapshot exclude=xxx** 后通常不需要在 **mass** 中重复写 **exclude=xxx**。

该机制适合收集无法提前精确列出名称的后处理产物。例如 Multiwfn、绘图脚本或外部程序在当前目录生成多种结果文件时, 可先执行 **snapshot**, 再运行后处理, 最后用 **mass(result_dir)** 统一移动新增产物。

4.2.3.8 keyword

keyword 用于在 **%preprocess** / **%process** 中就地编辑当前 workflow 正在处理的输入文件。它会根据当前任务的 **%program** 采用 Gaussian、ORCA 或 GAMESS 对应的关键词语法:

该指令始终作用于当前 block 的实际输入文件；对于 Gaussian 的关键词展开、多输入变体等场景，也会编辑当前正在执行的那一个 **.gjf** / **.inp**。

4.2.3.8.1 Gaussian 语法

```
keyword add td
keyword add td.nstate 5
keyword add freq.noraman true
keyword mod opt.recalc 3
keyword rm scrf
keyword rm scrf.smd
```

规则如下:

- **keyword add <kw>**：添加整个 route keyword，例如 **td**。
- **keyword add <kw>.<var> <value>**：添加 **kw(var=value)**。
- 当 **<value>** 为 **true**（大小写不敏感）或省略时，按布尔变量处理，例如 **freq.noraman true** -> **freq(noraman)**。
- **keyword mod <kw>.<var> <value>**：覆盖写入该变量；目标 keyword 或变量不存在时会补建。
- **keyword rm <kw>**：删除整个 keyword。
- **keyword rm <kw>.<var>**：只删除 keyword 内的一个变量。

4.2.3.8.2 ORCA 语法

```
keyword add TightSCF
keyword rm CPCM(Water)
keyword add %scf
keyword mod %pal.nprocs 16
keyword add %maxcore.value 3000
keyword add %scf:soscf
keyword mod %scf:soscf.start 0.002
keyword rm %scf:soscf.start
keyword mod %basis:newgto[1] C "def2-TZVP"
```

```
keyword rm %basis:newgto[1]
```

规则如下：

- **不带 % 前缀**的路径表示 ORCA ! 行上的 simple input keyword，例如 TightSCF、CPCM(Water)。
- keyword add <kw> / keyword mod <kw>：确保某个 ! keyword 存在；mod 对 simple keyword 等价于幂等的 add。
- keyword rm <kw>：删除 ! 行上的某个 keyword；缺失时静默跳过，workflow 继续执行。
- %input_block：表示顶层 %... input block 本身。例如 keyword add %scf 会确保 %scf block 存在，keyword rm %scf 会删除整个 block。
- %input_block.parameter：表示顶层 input block 内的 parameter。
  - 例如 keyword mod %pal.nprocs 16表示向%pal内加入nprocs 16。
- single-value block（如 %maxcore 3000）使用 %maxcore.value 表示其 value 字段。
- %parent:child：表示某个 child input block 本身。例如 keyword add %scf:soscf 会追加一个新的 soscf child block。
- %parent:child.parameter：表示 child input block 内的 parameter，例如 keyword mod %scf:soscf.start 0.002。
- %parent.child.parameter 也接受，语义与 %parent:child.parameter 相同；这里只是把 child 从路径中显式展开出来。
- child input block 可写 occurrence，下标从 0 开始，例如 %basis:newgto[1] 表示第二个 newgto child block。
- keyword add %parent:child ... 总是**追加**一个新的 child block；修改已有 occurrence 的 header tail 使用 keyword mod %parent:child[N] <header...>。
- ORCA block parameter 允许空值；因此 keyword add %geom.someflag 这类无显式 value 的写法会按 flag-style parameter 处理。

4.2.3.8.3 GAMESS 语法

```
keyword add SYSTEM.MWORDS 375
keyword mod CONICL.IXROOT(1) 1,2
keyword rm STATPT.OPTTOL
```

规则如下：

- 路径固定为 SECTION.KEY 形式；SECTION 可写成 SYSTEM 或 $SYSTEM。
- keyword add <section>.<key> <value>：若该 key 不存在则添加；若 section 不存在会先创建。
- keyword mod <section>.<key> <value>：覆盖写入；若 section/key 不存在则自动补建。
- keyword rm <section>.<key>：仅在该 key 存在时删除；缺失时静默跳过，workflow 继续执行。

4.2.3.9 result

result 用于向 .bane.result.user.kv 追加用户自定义结果，供 btproc result 在任务收尾时并入 .bane.result.kv 和 records/*.json。

支持以下形式：

```
result key value
result -k key -v value words
result -k SCF --exec grep "SCF Done" final.log
```

规则如下：

- 文字值会直接以 `key=value` 形式追加。
- `--exec` 会执行命令并捕获标准输出。捕获结果中的换行会被压平成空格。

4.2.3.10 dbcollect

`dbcollect` 用于向 `.bane.dbcollect.user.kv` 追加“只进入 record / database、不进入 `.bane.result.kv` 快照”的用户结果。它适合记录原子级、片段级、轨迹帧或其他不希望污染轻量级 `%when` 快照命名空间的数据。任务收尾时，`btproc result` 会通过 `--record-kv` 合并这些内容。

支持以下形式：

```
dbcollect key value
dbcollect -k key -v value words
dbcollect -k key --exec grep "Total charge" analysis.txt
dbcollect --merge-kv extra_values.kv
dbcollect --exec-kv python export_values.py
dbcollect --extxyz traj.extxyz --frame last
```

规则如下：

- `key value` / `-k key -v value` 会按 `key=value` 追加到 `.bane.dbcollect.user.kv`。
- `--exec` 捕获命令标准输出，作为指定 key 的值写入。
- `--merge-kv <file>` 直接合并已有 kv 文件。
- `--exec-kv <command...>` 把命令标准输出视为多行 kv 内容追加。
- `--extxyz <file> [--frame first|last|N]` 会调用 `btproc result extxyz-kv`，把 extxyz 中的 per-atom 属性转换为 `atoms.N.prop=val` 风格的 kv 行；未指定 frame 时默认使用最后一帧。

4.2.3.11 atomvec

`atomvec` 用于在 `%preprocess` / `%process` 中从上游任务读取 per-atom 数值向量，执行逐原子向量运算，并把结果写入当前任务的运行期 atom values。该命令面向通用 atom-level 数据派生，典型用途包括 RESP2 电荷平均、不同电荷方案之间的差值、spin density 变换、向量裁剪以及总量归一化。

基本形式如下：

```
atomvec <target_prop> = <expr> [options]
```

示例：

```
%process
  atomvec charge.resp2 = mean(resp_scrf:charge.resp, resp_vacuum:charge.resp)
↪   --check same_atoms --sum @resp_scrf.charge --out resp2.extxyz
```

该命令会生成一条 `btproc atomvec eval` 调用，并固定写入以下输出：

```
btproc atomvec eval --values-out .bane.values.kv --record-kv
↪  .bane.dbcollect.user.kv --snapshot-kv .bane.result.kv --dsl '<target_prop>
↪  = <expr> [options]'
```

执行后，目标向量会以 `atoms.N.<target_prop>=<value>` 的形式追加到 `.bane.values.kv`；同一组 atom values 也会追加到 `.bane.dbcollect.user.kv`，供 `btproc result` 写入 record / database。纯派生任务还会获得最小 `.bane.result.kv`，其中包含 `status=done`、`atom_count` 与 `geometry.atom_count`。

4.2.3.11.1 向量引用

向量引用用于读取某个任务目录中的 atom-level values。支持两种等价写法：

```
<task>:<prop>
<task>.atoms[].<prop>
```

读取规则如下：

- `<task>:<prop>` 会读取上游任务 `<task>` 中的 `atoms.1.<prop>`、`atoms.2.<prop>`、`atoms.3.<prop>` 等连续键。
- `<task>` 可以是依赖任务名，也可以是 `.` / `self` / `current`，后者表示当前任务目录。
- 任务目录解析顺序为当前任务目录的父目录下同名任务、当前任务目录下同名子目录、字面路径。
- 读取时会依次合并目标任务目录中的 `.bane.result.kv`、`.bane.values.kv`、`.bane.dbcollect.user.kv`、`.bane.values.user.kv`。
- 当存在 `atom_count` 或 `geometry.atom_count` 时，向量长度必须与该 atom count 一致。
- 目标属性 `<target_prop>` 会清理 `atoms[].` / `atoms.` 前缀；属性名中不能包含空白、换行或 `=`。

4.2.3.11.2 向量函数

`<expr>` 可以是单个向量引用，也可以是函数调用。函数可以嵌套，逗号只在顶层分隔参数。

- `mean(v1, v2, ...)`：接受一个或多个向量，对多个等长向量逐原子求平均。该函数用于 RESP2 中溶剂与真空 RESP 电荷的平均，也可用于多次 population 分析的平均。
- `add(v1, v2, ...)`：接受一个或多个向量，对多个等长向量逐原子求和。该函数用于合并多个 per-atom 贡献项。
- `sub(v1, v2)`：接受两个向量，逐原子计算 `v1 - v2`。该函数用于比较两种电荷方案、两种 spin population 或两次分析结果的差值。
- `scale(v, s)`：接受一个向量和一个标量，将向量每个元素乘以标量 `s`。该函数用于单位换算或线性缩放。
- `mul(v, s)`：`scale(v, s)` 的同义函数。
- `neg(v)`：接受一个向量，将向量每个元素取相反数。该函数用于反转差值方向。
- `abs(v)`：接受一个向量，将向量每个元素取绝对值。该函数用于得到逐原子差值幅度或 spin density 幅度。
- `clip(v, lo, hi)`：接受一个向量、下限和上限，将每个元素限制在 `[lo, hi]` 区间内；`lo` 不得大于 `hi`。该函数用于截断极端 population、权重或派生指标。
- `renorm(v, s)`：接受一个向量和目标总和，通过对所有元素施加同一个平移量，使向量总和等于 `s`。该函数用于将原子电荷总和校正为分子总电荷。
- `renormalize(v, s)`：`renorm(v, s)` 的同义函数。

等长要求由具体函数强制执行。**mean**、**add**、**sub** 需要参与运算的向量长度一致；**scale**、**mul**、**neg**、**abs**、**clip**、**renorm** / **renormalize** 保持输入向量长度不变。

4.2.3.11.3 标量表达式

标量表达式用于 **scale**、**mul**、**clip**、**renorm**、**--sum** 等位置，支持以下形式：

- 数字字面量：说明：有限实数，按 C 浮点规则解析；示例：**1.0**、**-0.5**、**2e-3**。
- **@key**：说明：读取当前任务目录合并后的 scalar value；示例：**@charge**。
- **@task.key**：说明：读取任务 **task** 的 scalar value；示例：**@resp_scrf.charge**。
- **sum(vector)**：说明：对一个向量表达式求总和，返回标量；示例：**sum(resp:charge.resp)**。

示例：

```
atomvec charge.delta = sub(scrf:charge.resp, vacuum:charge.resp) --check
↪  same_atoms
atomvec charge.scaled = scale(resp:charge.resp, -1.0)
atomvec charge.abs_delta = abs(sub(scrf:charge.resp, vacuum:charge.resp))
↪  --check same_atoms
atomvec charge.clipped = clip(resp:charge.resp, -0.8, 0.8)
atomvec charge.norm = renorm(resp:charge.resp, @resp.charge)
```

4.2.3.11.4 选项

- **--check same_atoms**：检查参与运算的向量与输出向量长度一致；当任务目录中可读取 **last_*.xyz**、***.xyz** 或 ***.extxyz** 时，还会检查元素序列一致。
- **--sum <scalar>**：在表达式求值后执行总量归一化，使结果向量总和等于 **<scalar>**。
- **--renorm <scalar>** / **--renormalize <scalar>**：**--sum** 的同义选项。
- **--out <file>**：将结果向量连同几何写出为 extxyz 文件。
- **--extxyz <file>**：**--out** 的同义选项。
- **--chg <file>**：将结果向量与几何写为 CHG 文本，每行包含元素、坐标和派生值。可与 extxyz 同时输出。
- **--format extxyz**：指定 extxyz 输出格式；只接受 **extxyz**。
- **--geom <task>**：指定 extxyz / CHG 使用哪个任务目录中的几何；省略、写 **first** 或 **auto** 时使用表达式中第一个向量来源，没有向量来源时使用当前目录。

4.2.3.11.5 extxyz 导出

启用 **--out** / **--extxyz** / **--chg** 时，**atomvec** 会从几何来源任务目录中读取几何。几何查找顺序为：

1. 文件名以 **last_** 开头、后缀为 **.xyz** 的文件；
2. 其他 **.xyz** 文件；
3. 其他 **.extxyz** 文件。

导出的 extxyz 使用以下属性布局：

```
Properties=species:S:1:pos:R:3:<target_prop>:R:1 generated_by=btvec
```

若 **<target_prop>** 含有 extxyz 属性名不允许的字符，导出时会把空白、**=**、**:** 等字符替换为 **_**；**.bane.values.kv** 与 record values 中仍使用原始 **<target_prop>**。

4.2.3.12 banewfn

```
banewfn density.fchk density.fchk
banewfn soc density.fchk --extra-flag
```

行为如下：

- 第一参数为模板基名。
- 第二参数为输入波函数文件。
- 查找顺序如下：
  1. 当前目录 **<base>.bwc**
  2. 当前目录 **<base>.inp**
  3. **BANETASK_BANEWFN_PATH** 路径列表中每个目录下的 **<base>.bwc**
  4. **BANETASK_BANEWFN_PATH** 路径列表中每个目录下的 **<base>.inp**
- 自动附加核心数参数 **-c ${CORES}** 或 **%CORES%**。

4.2.3.13 benv

```
benv g16
benv orca
```

行为如下：

- 从 **BANETASK_ENVCONF_PATH** 读取 **<name>.conf**；若该配置是路径列表，则按配置顺序使用第一个存在的配置文件。
- 取出该文件中的 **ENV_SETUP**，并写入当前命令脚本。
- 适用于在 **%preprocess** / **%process** 中切换额外程序环境。

4.2.3.14 fork

fork 是 **%process** / **%preprocess** 中的命令 DSL，用来把多帧 XYZ 或程序输出轨迹采集到 case 根目录侧的 **extra.yaml**。基础语法如下：

```
fork <input> as <collection_name> [and|, <item> ...]
fork conv(<output_file>) as <collection_name> [and|, <item> ...]
```

其中 **<item>** 可以是：

```
const(<value>) as <alias>
xyzmeta
xyzmeta as <namespace>
xyzmeta(<field> [as <alias>], ...)
xyzmeta(<field> [as <alias>], ...) as <namespace>
dist(i,j) as <alias>
angle(i,j,k) as <alias>
dihedral(i,j,k,l) as <alias>
```

示例：

```
fork cluster.xyz as cluster and const(298.15) as temperature and
↪  const("B3LYP-D3 BJ") as method and dist(1,2) as r12 and angle(3,4,5) as
↪  a345 and dihedral(6,7,8,9) as d6789
fork cluster.xyz as cluster, xyzmeta(energy as E, Boltzmann_w as w, Cum) as
↪  meta, dist(1,2) as r12
```

```
fork conv(opt.log) as optscan and xyzmeta(energy as E) as meta and const(scan)
↪  as source and dist(1,2) as r12
```

行为如下：

- 基础形态为 **fork <input> as <collection_name>**；**and** 和逗号都可以作为后续 item 的分隔符。
- **conv(<output_file>)** 等价于对程序输出轨迹启用 **btproc fork collect --conv**，只收集已收敛的结构帧。
- **const(<value>) as <alias>** 会把同一个常量字段写入每一个集合元素；带空格的字符串需要加引号，例如 **const("B3LYP-D3 BJ") as method**。
- **xyzmeta** 会把可获得的 XYZ frame metadata 写入集合元素；**xyzmeta as meta** 会把这些字段放进 **meta:** 子映射。
- **xyzmeta(energy as E, Boltzmann_w as w, Cum)** 只写指定字段；字段后面的 **as <alias>** 是输出字段名，不写时输出名等于原字段名。再接 **as meta** 时，这些字段会写到 **meta:** 子映射里。
- **xyzmeta** 可用的内建字段包括 **comment**、**energy**、**maxForce**、**rmsForce**、**maxDisp**、**rmsDisp**、**charge**、**multiplicity**；XYZ 注释行中解析出的 **key=value** 元数据也可按原名读取，例如 **Boltzmann_w**、**Cum**。内建字段匹配不区分大小写，注释行元数据名按原样匹配。
- **dist(i,j)**、**angle(i,j,k)**、**dihedral(i,j,k,l)** 使用 1-based 原子编号。
- 展开时会根据输入路径和当前 **%program** 自动推断 **btproc fork collect** 模式：**.xyz** 对应 **xyz**，**.gms** 对应 **gms**；其余情况下优先跟随当前任务程序 **gaussian** / **orca** / **gms** / **mopac**，否则回退到 **auto**。
- 输出位置固定在 case 根目录侧的 **extra.yaml**，适合把任务目录中产生的多帧 XYZ 或程序输出轨迹收集成结构化集合，并预先写入后续 **%when** 需要的几何字段。
- 常见用法是与 **@foreach? cluster**、**%source xyz=cluster frame=[cluster.frame]**、**%when [cluster.r12] < 2.5** 配合，驱动后续的按帧任务展开与筛选。
- 由于 **@foreach** / **@foreach?** 的预展开发生在解析阶段，**fork** 在当前 workflow 中生成的 **extra.yaml** 会在下一次 **banetask** 刷新或重建时生效。

4.2.3.15 原样命令

未识别为 DSL 指令的命令行会按原样写入 shell 脚本，并执行占位符替换。例如：

```
grep "Normal termination" *.log
python plot.py [taskname]
```

4.2.4 自定义 Process DSL 插件

自定义插件可以给 **%preprocess** / **%process** 增加新的命令名，而不要求任务作者拼接平台相关的 shell 引号。插件由 JSON manifest 声明，输出结构化 argv 或一组文件、目录和 kv action。插件只在生成 **pre_comd** / **comd** 时展开，不在运行 workflow 时重新解析 manifest；因此生成后的脚本可以独立执行。

纯数值结果计算、单位换算、任务表和逐原子表连接应优先使用 Derive DSL。Process DSL 插件适合执行外部程序、操作文件、修改当前输入、生成 **extra.yaml**，或把运行期数据写入 result、record、values 和 snapshot 链路。

4.2.4.1 启用、搜索与冲突规则

自定义插件默认启用。相关配置如下：

配置项	默认行为	说明
`BANETASK_PROCESS_PLUGINS_ENABLED`	启用	设置为非真值时不发现或加载自定义 manifest；内建命令仍可用。
`BANETASK_PROCESS_PLUGIN_PATH`	空	追加搜索根或 manifest 文件。POSIX 可用 `:` 或 `;` 分隔，Windows 用 `;`；路径中的环境变量会展开。
`BANETASK_PROCESS_PLUGIN_ALLOW_OVERRIDE`	关闭	关闭时内建 alias 优先；开启时自定义 alias 可覆盖内建 alias。
`BANETASK_PROCESS_PLUGIN_STRICT`	关闭	开启后把发现和加载诊断记为 error；无效项仍被跳过，不会仅因该开关自动终止整个解析。
`BANETASK_CMD_ENABLE_DIRECTIVES`	启用	关闭时所有内建命令和自定义插件均停用，每一行直接进入 raw shell fallback。

搜索根按下列顺序加入：

1. 当前工作目录的 `.banetask/plugins`
2. `~/.bane/task/plugins`
3. `banetask` 可执行文件旁的 `plugins`
4. `BANETASK_PROCESS_PLUGIN_PATH` 中从左到右的条目

每个搜索根可以是以下任一种形式：

```
/path/to/plugin.json
/path/to/name.btplugin.json
/path/to/name.plugin.json
/path/to/plugin-directory/
/path/to/plugins-root/
```

目录扫描规则如下：

- 目录自身存在 `plugin.json` 时，该文件是一个 manifest，插件目录为该目录。
- 根目录的每个直接子目录若存在 `plugin.json`，也会被加载。
- 根目录第一层的 `*.btplugin.json`、`*.plugin.json` 和 `plugin.json` 会被识别。
- 不递归扫描任意深度；需要分组时，每个直接子目录放一个 `plugin.json`。
- `*.btplugin.json` 是单文件 manifest 的规范后缀；`*.plugin.json` 也可识别，并产生诊断。
- 候选 manifest 按完整路径排序并去重。两个自定义插件以相同优先级注册同一个 alias 时，排序后先加载的 manifest 生效。

alias 匹配区分大小写。内建命令名与自定义命令名使用同一个注册表；关闭全局覆盖时，任何自定义 command 对象都不能单独绕过内建优先级。

4.2.4.2 命令行拆分

插件收到三个命令视图：完整行、命令名和命令正文。常规写法以第一个空白分隔：

```
socplot soc.log --out soc.png
```

其中命令名是 **socplot**，正文是 **soc.log --out soc.png**。**name(args)** 写法也会把 **name** 识别为命令名，并把从左括号开始的文本保留在正文中：

```
mycmd(alpha, beta) --flag
```

正文还会按命令行引号规则拆成 token，供 **{args...}** 和 **{arg0}** 等模板使用。插件未命中时，整行按原样 shell 命令输出；命中插件但 manifest command 生成空 action 时，不再回退为 raw shell。

%preprocess / **%process** 原始块支持行尾单个反斜杠 **** 续行。续行发生在命令识别之前，可用于拆分较长的插件调用或 shell 命令。

4.2.4.3 manifest 顶层结构

manifest 顶层必须是 JSON object：

```
{
  "api": "banetask.process.plugin.v1",
  "name": "socplot",
  "version": "0.1.0",
  "commands": [
    {
      "id": "custom.socplot.render",
      "aliases": ["socplot"],
      "mode": "argv-template",
      "effects": ["emits_raw_shell"],
      "argv": ["python", "${plugin_dir}/socplot.py", "{args...}"]
    }
  ]
}
```

字段	要求	说明
api	必填 string	必须精确为 **banetask.process.plugin.v1**；其他值使整个 manifest 被忽略。
name	可选 string	插件名；默认取插件目录 basename。
version	可选 string	写入任务元数据的插件版本；默认 **0**。
commands	必填 array	每个 object 定义一个命令；非 object 元素被忽略。

每个 command object 支持：

字段	要求	说明
aliases	必填 string array	一个或多个命令名。空字符串会被移除；最终没有 alias 时跳过该 command。单个 string 字段 **alias** 也可接受，并产生诊断。

字段	要求	说明
`id`	可选 string	稳定命令标识。省略时，单命令 manifest 使用 `custom.<plugin-name>`；多命令 manifest 使用 `custom.<plugin-name>.<first-alias>`。
`mode`	可选 string	`argv-template`、`json-actions` 或 `actions`；默认 `argv-template`。`actions` 与 `json-actions` 等价。
`effects`	可选 string array	声明命令产生的副作用，详见“effect”。未知名称被忽略。
`argv`	`argv-template` 使用	argv 模板数组。
`actions`	`json-actions` 使用	action object 数组。

4.2.4.4 `argv-template`

`argv-template` 把一个 DSL 命令展开成一条结构化命令：

```
{
  "api": "banetask.process.plugin.v1",
  "name": "socplot",
  "version": "0.1.0",
  "commands": [
    {
      "aliases": ["socplot"],
      "mode": "argv-template",
      "argv": [
        "python",
        "${plugin_dir}/socplot.py",
        "{args...}",
        "--task",
        "{task.inputName}"
      ]
    }
  ]
}
```

任务文件：

```
%process
  socplot soc.log --out soc.png
```

宿主先展开模板，再按目标 shell 对每个 argv 项进行安全引用。`argv` 必须是数组；非 string 元素被忽略。展开后 argv 为空时，该命令只产生诊断，不输出 shell 行。

不要把多个 shell 参数拼成一个模板项。例如 `"--input {arg0}"` 会成为一个 argv 项；应写成 `"--input", "{arg0}"`。需要管道、重定向或 shell 控制运算符时，使用 `json-actions` 的 `raw` action，并显式声明 `emits_raw_shell`。

4.2.4.5 `json-actions`

`json-actions` 可按顺序生成多种动作：

```
{
  "api": "banetask.process.plugin.v1",
  "name": "custom-analysis",
  "commands": [
    {
      "aliases": ["analyze-and-record"],
      "mode": "json-actions",
      "actions": [
        {
          "op": "exec",
          "argv": ["python", "${plugin_dir}/analyze.py", "{arg0}", "--out",
          ↪  "analysis.json"]
        },
        {
          "op": "append_kv_exec",
          "target": "record",
          "key": "custom.score",
          "argv": ["python", "${plugin_dir}/read_score.py", "analysis.json"]
        },
        {
          "op": "copy",
          "source": "analysis.json",
          "destination": "../Results/{task.inputName}/analysis.json"
        }
      ]
    }
  ]
}
```

`actions` 必须是数组。非 object 元素、未知 `op`、缺少 action 必需字段或展开后缺少 argv 的 action 会产生诊断并被跳过；同一 command 中其他有效 action 继续生成。

`op`	字段	行为
`raw`	`line`	原样输出一条 shell 行，并自动加入 `emits_raw_shell` effect。空行不输出。
`comment`	`text`	按目标 shell 语法输出注释。
`exec`	`argv` 或 `command`	输出一条结构化 argv 命令；两个字段都存在时优先 `argv`。
`copy`	`source`、`destination`、可选 boolean `recursive`	复制文件；`recursive=true` 时按目录复制。默认 `false`。
`move`	`source`、`destination`	移动文件或目录。
`mkdir`	`path`	创建目录；按目标 shell 使用可重复执行的目录创建语义。

op	字段	行为
append_kv	target 或 file/kvFile、key、value	追加一条 key-value。
append_kv_exec	target 或 file/kvFile、key、argv/command	执行命令，把标准输出作为指定 key 的值追加。
merge_kv	target 或 file/kvFile、source	把一个 kv 文件的内容追加到目标 kv 文件。
stream_exec	target 或 file/kvFile、argv/command	执行命令，把其标准输出流直接追加到目标 kv 文件。
extxyz_kv	可选 target 或 kvFile、extxyz/source/path、可选 extraArgs	调用 btproc result extxyz-kv 并把输出追加到 kv。几何路径按 extxyz、source、path 的顺序取第一个非空值；target 默认 record。

argv、command 和 extraArgs 都是 argv 模板数组，遵守与 argv-template 相同的展开规则。file 与 kvFile 等价；显式文件路径可写任意目标。使用显式文件路径时，宿主不能从文件名推断 effect，必要时应在 command 的 effects 中显式声明。

4.2.4.6 kv target

下列 target 会映射到任务目录中的标准缓冲文件，并自动加入对应 effect：

target	实际文件	自动 effect
result、extra、extra-kv	.bane.result.user.kv	writes_result_kv
record、dbcollect、record-kv	.bane.dbcollect.user.kv	writes_record_kv
values、value、values-kv	.bane.values.user.kv	writes_values_kv

未知 target 且未给出 file/kvFile 时，该 action 被跳过。append_kv 等 action 只负责写入缓冲；最终是否进入轻量快照、规范 record 或运行期 values 取决于该任务是否启用了相应结果导出链路。

4.2.4.7 模板变量

模板可在 argv 项、action 字符串字段和路径中使用。未知变量、缺失 header/config 值和越界 {argN} 展开为空字符串。

变量	内容
${plugin_dir}、{plugin_dir}、{paths.pluginDir}	manifest 所属插件目录。
{command.raw}	去除首尾空白后的完整命令行。
{command.name}	已匹配的命令名。

变量	内容
{command.content}、{content}	命令名之后的原始正文；name(args) 中从左括号开始。
{args}	token 以单个空格连接后的字符串。它仍是一个 argv 项。
{args...}	仅当它是 argv 数组中的完整独立元素时，将每个 token 展开为独立 argv 项。嵌在其他文字中时不作多项展开。
{arg0}、{arg1}、…	第 N 个 token，从 0 开始。
{stage}	preprocess 或 process。
{task.outputDir}	当前任务输出目录。
{task.taskName}	当前任务名。
{task.rootName}	case 根名。
{task.originName}	原始输入基名。
{task.inputName}	<task>_<origin> 形式的输入基名。
{task.sourceName}	%source 的主来源名。
{task.strucId}	%meta strucid 解析出的结构标识；未设置时为空。
{task.program}、{task.programTypeName}	当前程序类型名。
{paths.scriptsPath}、{paths.BANETASK_SCRIPTS_PATH}	解析后的脚本路径配置。
{paths.wfnExamplesPath}、{paths.BANETASK_WFN_EXAMPLES_PATH}	解析后的 Multiwfn 模板路径配置。
{paths.externalScriptsPath}、{paths.BANETASK_EXTERNAL_SCRIPTS_PATH}	解析后的外部环境脚本路径配置。
{paths.baneWfnPath}、{paths.BANETASK_BANEWFN_PATH}	解析后的 banewfn 模板路径配置。
{config.<KEY>}	运行时配置中 <KEY> 的最终值，包括环境变量覆盖。
{header.<name>}	YAML 头部扁平化变量 <name> 的值。

展开顺序为：插件模板变量先展开，action 再转换为目标 shell 命令，最后执行普通任务运行时占位符替换。因此插件输出中仍可使用 [taskname]、[inputname]、[originname]、[rootname] 等命令脚本占位符。

4.2.4.8 effect

effects 是 command 对副作用的声明。正式名称及接受的短名如下：

正式名称	接受的短名	含义
`writes_result_kv`	`result`	写 `.bane.result.user.kv`。
`writes_record_kv`	`writes_dbcollect_kv`、`record`、`dbcollect`	写 `.bane.dbcollect.user.kv`。
`writes_values_kv`	`values`	写 `.bane.values.user.kv` 或运行期 values。
`writes_snapshot_kv`	`snapshot`	写轻量结果快照。
`modifies_current_input`	`keyword`	修改当前量化程序输入。
`writes_extra_yaml`	`extra_yaml`	写 case 级 `extra.yaml` 集合。
`requires_external_env`	`env`	依赖外部程序环境。
`emits_raw_shell`	`raw_shell`	输出任意 shell 文本。

`writes_result_kv`、`writes_record_kv` 或 `writes_values_kv` 中任一项存在时，生成的命令脚本会在插件动作之前清理旧的 user result、record、values 缓冲和 `.bane.values.kv`，防止上次运行残留被再次合并。使用标准 kv target 的 action 会自动添加这三类 effect。

其他 effect 本身不创建文件、不执行命令，也不会替代 action；插件必须实际生成对应操作。它们用于准确声明副作用并写入生成期诊断。不要仅声明 `writes_snapshot_kv` 就假定宿主会自动生成 snapshot。

4.2.4.9 诊断、元数据与可复现性

- manifest 无法打开、JSON 无法解析、API 不匹配或缺少 `commands` 时，整个 manifest 被忽略。
- command 无 alias、action 无必要字段或 mode 不受支持时，只跳过对应层级。
- `BANETASK_PROCESS_PLUGIN_STRICT=true` 只改变诊断级别，不改变上述跳过和继续规则。
- 已解析的插件调用会在 `.bane.taskmeta.json` 的 `processPlugins` 数组中记录 `id`、`version` 和 `builtin`。同一 `id` + `version` 在一个任务中只记录一次。
- raw shell fallback 也以 `builtin.raw_shell` 记录。由此可以区分结构化插件命令和未识别 shell 行。
- 任务元数据记录插件标识，不内嵌 manifest 或插件程序。需要长期复现时，应同时封存插件目录、版本及其外部依赖。

完整自定义示例：

```
{
  "api": "banetask.process.plugin.v1",
  "name": "charges",
  "version": "1.2.0",
```

```
  "commands": [
    {
      "id": "custom.charges.resp",
      "aliases": ["resp-export"],
      "mode": "json-actions",
      "effects": ["requires_external_env"],
      "actions": [
        {
          "op": "exec",
          "argv": ["python", "${plugin_dir}/resp.py", "{arg0}", "--output",
          ↪  "resp.extxyz"]
        },
        {
          "op": "extxyz_kv",
          "target": "record",
          "extxyz": "resp.extxyz",
          "extraArgs": ["--frame", "last"]
        },
        {
          "op": "copy",
          "source": "resp.extxyz",
          "destination": "../Results/{task.inputName}/resp.extxyz"
        }
      ]
    }
  ]
}
```

```
$resp
  %source sp
  %process
    resp-export [inputname].log
```

4.2.5 生成的命令脚本环境变量

pre_comd 与 **comd** 会自动导出以下环境变量：

- **BANETASK_TASKNAME**：当前任务名。
- **BANETASK_ORIGINNAME**：原始输入基名。
- **BANETASK_INPUTNAME**：**<task>_<origin>**。
- **BANETASK_RUNID**：与 **BANETASK_INPUTNAME** 相同。
- **BANETASK_ROOTNAME**：当前 case 根名。
- **BANETASK_SOURCENAME**：**%source** 值。
- **BANETASK_DEPENDENCIES**：运行时依赖列表，逗号分隔。
- **BANETASK_STRUCID**：**%meta strucid** 指定的结构标识。
- **BANEVAR_YAMLFILE**：../var.yaml。
- **BANETASK_SCRIPTS_PATH**：命令 DSL 中 **run/scripts** 的脚本目录或路径列表。
- **BANETASK_WFN_EXAMPLES_PATH**：命令 DSL 中 **mwfn/multiwfn** 的模板目录或路径列表。

- `BANETASK_BANEWFN_PATH`：命令 DSL 中 `banewfn` 的模板回退目录或路径列表。

4.2.5.1 实际输入、输出与命令模板变量

workflow 先加载 `BANETASK_EXTERNAL_SCRIPTS_PATH` 指向的外部环境，再执行 `ENV_SETUP`，之后在 `pre_comd` 之前导出实际文件上下文。`RUN_CMD_TEMPLATE`、`pre_comd` 和 `comd` 可使用这些变量：

变量	含义
`BANETASK_ACTUAL_INPUT_FILE`	实际输入路径或文件名；是否含绝对路径取决于生成配置。
`BANETASK_ACTUAL_INPUT_FILENAME`	不含目录的实际输入文件名。
`BANETASK_ACTUAL_INPUT_STEM`	去掉最后一个后缀的实际输入 stem。
`BANETASK_ACTUAL_OUTPUT_FILE`	实际输出路径或文件名。
`BANETASK_ACTUAL_OUTPUT_FILENAME`	不含目录的实际输出文件名。
`BANETASK_ACTUAL_OUTPUT_STEM`	去掉最后一个后缀的实际输出 stem。
`ACTUAL_INPUT_FILE`、`ACTUAL_INPUT_FILENAME`、`ACTUAL_INPUT_STEM`	对应 `BANETASK_ACTUAL_INPUT_*` 的简写别名。
`ACTUAL_OUTPUT_FILE`、`ACTUAL_OUTPUT_FILENAME`、`ACTUAL_OUTPUT_STEM`	对应 `BANETASK_ACTUAL_OUTPUT_*` 的简写别名。
`FILE_NAME`、`BASENAME`	`BANETASK_ACTUAL_INPUT_STEM` 的别名。

`RUN_CMD_TEMPLATE` 中的 `${ACTUAL_INPUT_FILE}`、`${ACTUAL_OUTPUT_FILE}`、`${ACTUAL_INPUT_FILENAME}`、`${ACTUAL_OUTPUT_FILENAME}`、`${ACTUAL_INPUT_STEM}`、`${ACTUAL_OUTPUT_STEM}`、`${FILE_NAME}` 和 `${BASENAME}` 使用同一组值。新配置宜使用 `ACTUAL_*` 名称；`FILE_NAME` 与 `BASENAME` 主要用于接受已有模板。

4.2.5.2 阶段状态变量

生成的 workflow 用以下变量把执行结果传给后续阶段：

变量	可用时机与含义
`BANETASK_MAIN_RC`	主命令结束后可用，保存主命令的原始返回码。
`BANETASK_MAIN_OK`	主命令结束后可用。值为 `1` 表示主阶段按任务策略成功，值为 `0` 表示失败；当 `%control alert false` 时，非零 `BANETASK_MAIN_RC` 仍可对应 `BANETASK_MAIN_OK=1`。
`BANETASK_PREPROCESS_RC`	`pre_comd` 结束后可用，保存整个前处理块的最终返回码；没有前处理块时不保证定义。
`BANETASK_PREPROCESS_OK`	workflow 启动时为 `1`；前处理块结束后按最终返回码更新为 `1` 或 `0`。
`BANETASK_PROCESS_RC`	`comd` 结束后保存整个后处理块的最终返回码；不能在同一个 `comd` 块内部读取其最终值。

变量	可用时机与含义
`BANETASK_PROCESS_OK`	workflow 启动时为 `1`；后处理块结束后按最终返回码更新为 `1` 或 `0`。

Bash 的 `pre_comd` 和 `comd` 采用软块执行：块内中间命令失败不会自动中止，阶段返回码取块内最后执行命令的返回码。需要任一命令失败即使阶段失败时，应在自定义脚本中显式使用 `set -e`、条件判断或累计返回码。

4.2.5.3 `btrun` 分配变量

由 `btrun` 启动 workflow 时，还会提供资源和作业上下文：

变量	含义
`CORES`	本次任务获得的有效 CPU 核心数。
`BTRUN_JOBID`	调度系统返回的原生作业 ID；本地执行或尚无原生 ID 时使用 btrun 作业名。
`BTRUN_GPUS`、`BANETASK_GPUS`	分配到的 GPU 数量。
`BTRUN_GPU_IDS`、`BANETASK_GPU_IDS`	可确定设备编号时的 GPU ID 列表；无设备分配时可能未定义。
`CUDA_VISIBLE_DEVICES`、`ROCR_VISIBLE_DEVICES`	本地队列或调度包装器可确定 GPU ID 时，限制为本次分配的设备列表。

调度包装器还可设置 `BTRUN_LOCAL_CONTEXT`、`BTRUN_TOTAL_CORES`、`BTRUN_TOTAL_MEMORY_MB` 和 `BTRUN_TOTAL_GPUS`，供 workflow 内嵌套本地队列判断可用资源。它们属于 runner 上下文，不应在任务脚本中手工伪造。直接执行 `.bwrk` 时，只有调用方显式提供的 `btrun` 分配变量才存在。

workflow 初始化时，若 `BANETASK_EXTERNAL_SCRIPTS_PATH` 路径列表中的目录存在以下文件，则会在 `ENV_SETUP`、`pre_comd`、主命令和 `comd` 之前按目录顺序加载第一个命中的脚本；即使任务没有前后处理命令也执行该步骤：

- Bash：`env.sh`
- Cmd：`env.bat`、`env.cmd`

4.2.5.4 任务目录 `.env` 与后处理重跑

使用 Bash workflow 且任务目录中存在 `comd` 时，workflow 会在进入 `comd` 前把当时可见的后处理环境合并写入任务目录 `.env`。即使 `.bwrk` 已被 `btrun` 消费并删除，`.env`、`comd` 和主程序输出仍可共同用于重新执行后处理。

`.env` 同时容纳三类内容：

- 用户自行编写的内容。自动更新时原样保留。
- `# >>> btrun runtime environment >>>` 与对应结束标记之间的运行时区块，记录本次分配的 `CORES`、作业标识、GPU 数量、GPU ID 和设备可见性等重跑所需变量。调度内部使用的 `BTRUN_LOCAL_CONTEXT` 与 `BTRUN_TOTAL_*` 不写入该区块，避免人工加载后改变后续任务的路由语义。

- `# >>> banetask postprocess environment >>>` 与对应结束标记之间的后处理区块，记录实际输入/输出文件上下文、任务阶段状态、BaneTask/btrun 变量，以及外部环境初始化、`ENV_SETUP` 或 `pre_comd` 新增或改写的导出变量。

两个自动区块由程序维护，不应手工编辑。每一方更新时只替换自己的区块，不会删除用户内容或另一方的区块。多个 workflow 共用同一任务目录时，后处理区块表示最近一次进入 `comd` 的 workflow。

自定义环境变量需要在 `ENV_SETUP` 或 `pre_comd` 中使用 `export` 才会自动进入后处理快照。还可设置 `BANETASK_ENV_PERSIST_VARS`，用逗号或空格列出需要强制保存的额外变量名，例如：

```
export BANETASK_ENV_PERSIST_VARS="MULTIWFN_PATH,MY_POSTPROC_HOME"
```

默认文件为任务目录 `.env`；需要改名或放到其他位置时，可在提交环境中设置 `BANETASK_ENV_FILE`。相对路径分别以实际 job 工作目录和 workflow 工作目录解析，因此通常应使用绝对路径，或继续采用默认值。这个控制变量只用于定位文件，不写入自动区块，避免 remote profile 拉回 `.env` 后残留远端绝对路径。

在任务目录中重新执行 `comd`：

```
(
    set -a
    . ./.env
    set +a
    bash ./comd
)
```

`.env` 使用 Bash 可读取的赋值、`export` 与 `unset` 语句，自动更新时权限收紧为仅文件所有者可读写。环境初始化命令可能把路径、令牌或其他敏感值写入该文件，应按任务输出中的敏感文件管理。该机制适用于 Bash workflow；Windows Cmd workflow 不生成或读取此 `.env` 快照。

4.3 Workflow 与执行产物

在 BaneTask 中，输入文件只是任务产物的一部分。真正参与执行调度的，还包括前后处理脚本、workflow 文件、状态记录以及若干 sidecar 文件。这些内容共同构成了一个任务从“描述”走向“可执行”的完整输出集合。

4.3.1 `pre_comd` 与 `comd`

`pre_comd` 与 `comd` 分别对应主程序执行前后的实际可执行脚本文件。前者通常承载准备目录、复制依赖文件和切换环境等动作，后者则更常用于归档结果、执行后处理以及提取额外指标。当任务包含 `result` DSL 或启用结果快照导出时，系统还会在前置阶段清理旧的 `.bane.result.user.kv`，以避免残留内容污染本次结果。

内置 `comd.conf` 使用 `INPUT_FILENAME=comd`，因此可以通过 `btrun task -t comd <目录>` 精确发现并提交这些无后缀脚本。`comd` 或 `pre_comd` 本身作为主输入时，`btrun` 不再自动读取和串接同目录的任何 command sidecar；主输入只执行一次，避免后处理递归执行或前处理被意外重放。

4.3.2 `.bwrk` workflow

每个输入文件都会对应一个 `.bwrk` workflow 文件。它是把环境初始化、前处理命令、主程序调用和后处理步骤串联起来的执行载体，也是外部调度系统或 runner 实际消费的核心文件之一。主要消费端是 `btrun submit` / `btrun render` / `btrun kick`。其内容结构如下：

1. YAML 头部
2. `ENV_SETUP`
3. `pre_comd` 内容
4. 展开后的 `RUN_CMD_TEMPLATE`
5. 主命令返回码捕获
6. Bash workflow 合并更新任务目录 `.env` 的后处理区块
7. `comd` 内容
8. 在阶段状态允许时调用独立的 `autorun_btrun` / `autorun_btrun.bat`
9. 以主命令与 workflow 状态规则计算最终返回码；autorun 自身返回码只记录日志

典型头部如下：

```
prog: gaussian
nprocs: 32
memory: 96000
---
```

或：

```
prog: orca
nprocs: 32
maxcore: 3000
---
```

4.3.2.1 `RUN_CMD_TEMPLATE` 可用变量

环境配置文件中的 `RUN_CMD_TEMPLATE` 与 `btrun task --post-var NAME` 指向的配置变量支持 `${NAME}` 与 `$NAME` 两种占位符形式；建议新配置统一使用 `${NAME}`，避免 `$NAME_SUFFIX` 一类字符串产生歧义。可用变量包括：

- `${ACTUAL_INPUT_FILE}`：当前输入文件路径。默认使用完整路径；若 `<type>.conf` 中设置 `USE_ABSOLUTE_PATH=false`，则使用输入文件名。
- `${ACTUAL_OUTPUT_FILE}`：当前输出文件路径。默认由输入文件名与 `OUTPUT_SUFFIX` 推导。
- `${ACTUAL_INPUT_FILENAME}`：输入文件名，不含目录。
- `${ACTUAL_OUTPUT_FILENAME}`：输出文件名，不含目录。
- `${ACTUAL_INPUT_STEM}`：输入文件名去后缀。
- `${FILE_NAME}` / `${BASENAME}`：当前任务基础名，通常等于输入文件 stem。
- `${WORKDIR}`：当前 job 的工作目录。
- `${TYPE}`：当前 task type，即解析到的 `<type>.conf` 的 type 名。
- `${CORES}`：当前任务实际核心数。
- `${EXTRA_ARGS}` / `${ARGS}`：当前任务的程序参数；优先来自 `btrun task --extra-args`，否则来自 `task.manifest.json` 中的 `program_params.args` 或 `program_params.extra_args`。

示例：

```
[RUN_CMD_TEMPLATE]
g16 "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}"
```

4.3.2.2 主命令状态变量

workflow 在执行主程序后会导出或设置：

- **BANETASK_MAIN_RC**：主命令返回码
- **BANETASK_MAIN_OK**：主命令是否成功；成功为 **1**，失败为 **0**

4.3.3 bt.status

bt.status 用于记录任务块最近一次生成阶段的状态。新格式是面向人工阅读的四列表格：

```
# TASK            STATUS          UPDATED_AT              REASON
opt               RENDERED        2026-07-03T10:31:40Z    -
freq              FAILED          2026-07-03T10:31:42Z    workflow_write_failed
```

STATUS 目前包含两类：**RENDERED** 表示该任务块的输入、命令脚本与 workflow 已成功生成；**FAILED** 表示上一次生成阶段失败，**REASON** 给出简短原因。

banetask --comd-only 用于刷新 **pre_comd** 和 **comd**；**.bwrk** 与 **bt.status** 保持原状态，并且只对 **bt.status** 中处于 **RENDERED** 状态的任务块生效。**btool project update** 正是基于这一模式工作，而 **btool project rebuild** 则会清理选定任务块的状态记录，重新生成输入与 workflow，并同步写入新的 **RENDERED** 或 **FAILED** 记录。

当头部启用了 **finalize** 时，系统还可能在所有有效任务分支完成后额外写入一个隐式收尾任务名（默认 **__banetask_finalize__**），并在对应目录下生成一套 script/workflow 产物；若计划重新变为未完成状态，该记录会被自动清理。

4.3.4 任务目录中的文件

任务目录中的文件既反映生成阶段的输出，也反映结果链路是否已经建立。理解这些文件的用途，有助于在排查生成问题、执行问题或结果同步问题时快速定位故障点。

对于 **%keywords** 关键词集合展开产生的多输入变体，以下 **<task>_<origin>** 前还可能额外带有由关键词组合生成的前缀；本文仅以未展开时的基本命名表示。

- **<task>_<origin>.gjf**（文本）：Gaussian 输入文件。
- **<origin>_<task>.qc2g16_external.conf** 与 **<origin>_<task>.external.sh** / **<origin>_<task>.external.cmd**（文本）：Gaussian 任务通过 **%interface consume** 接入 Gaussian、XTB、ORCA、GAMESS 或 AMESP provider 时生成的 **qc2g16_external** 配置与 External 启动脚本；Windows 使用 **.cmd**，其他平台使用 **.sh**。
- **<task>_<origin>.inp**（文本）：ORCA、GAMESS、BDF 或 MAPLE 输入文件。
- **MINP**（文本）：MRCC 固定文件名输入；task manifest 与 workflow 均直接记录该名称，不再生成或复制 **<task>_<origin>.inp**。
- **<task>_<origin>.mop**（文本）：MOPAC 托管输入文件；同名 **.out** 是主输出，启用 **AUX** 时同名 **.aux** 由 MOPAC 原生写出。

- **<task>_<origin>.molden.input**（文本）：ORCA 任务在 **<task>_<origin>.gbw** 存在时由自动 **orca_2mkl <task>_<origin> -molden** 后处理生成。
- **<task>_<origin>.<suffix>**(文本):**other** 模板生成的输入文件,后缀由模板或 **%param suffix=...** 决定。
- **<task>_<origin>.sh** / **<task>_<origin>.bat**（文本）：**script** 程序生成的 launcher 脚本。
- **<task>_<origin>.payload.<ext>**（文本）：**script** runner 需要 payload 包装时生成的脚本正文文件。
- **.batch/frame_*/input.xyz**（文本）：批处理启用 **%batch** 后为每个 frame 生成的独立输入几何。
- **.batch/frame_*/run.sh**（文本）：当前 frame 的实际运行脚本。
- **.batch/frame_*/run.log**（文本）：当前 frame 的标准输出和标准错误日志。
- **.batch/frame_*/<input-stem>.out** / **.aux**（文本）：MOPAC batch 的原生主输出与可选 AUX；**run.log** 只保存启动器输出，避免覆盖同 stem **.out**。
- **.batch/frame_*/status.json**（JSON）：当前 frame 的运行状态。
- **.batch/frame_*/run.log** 与 **.batch/frame_*/status.json**（文本/JSON）：XTB 批处理每个 frame 的运行日志和状态文件。
- **.batch/manifest.json** 与 **.batch/run_batch.sh**（JSON/文本）：批处理的内部 frame manifest 与父 workflow 调用的 batch driver；manifest 可包含 **collect_rules** 和 **cluster** 配置。
- **final/batch.summary.json**、**final/failed_frames.txt**（JSON/文本）：批处理的成功/失败 frame 数、轨迹收集数和可选聚类状态。
- **final/optimized_traj.xyz** 与 **final/optimized_traj.frames.txt**（文本）：成功提取的优化结构轨迹及其原 **frame_XXXXXX** 映射。
- **final/clustered_traj.xyz** 与 **final/cluster.summary.json**（文本/JSON）：请求 **%batch cluster** 且聚类成功时生成的代表结构轨迹和完整簇成员报告。
- **<task>_<origin>.xyz**（文本）：当前任务物化的主几何，也是 XTB、**other** 和需要 XYZ 的 script 任务所使用的标准化坐标文件。
- **<task>_<origin>.<slot>.xyz**（文本）：由多行 **%source** 的 **geom.<slot>** 物化的命名辅助几何。
- **<xyz-file>**（文本）：由 **%control finalgeom <xyz-file>** 声明的用户产出最终几何；BaneTask 不主动生成它，但会在下游 **%source <task>**、运行期依赖检查和 **completed** 条件中把它作为几何来源与完成标志。
- **<task>_<origin>.xcontrol**（文本）：XTB 在存在 **%extrainput** 或 **%mod freeze** 时写出的 xcontrol sidecar。
- **pre_comd**（文本）：前处理脚本。
- **comd**（文本）：后处理脚本。
- **autorun_btrun** / **autorun_btrun.bat**（文本）：头部 **autorun** 为 **strict**、**default**、**true** 或 **always** 时生成的独立续扫脚本。
- **.banetask_autorun_btrun.trace**(文本): 独立 autorun 调用的默认日志; 可由 **BANETASK_AUTORUN_LOG** 改写。
- **<task>_<origin>.bwrk**（文本）：workflow 文件。
- **<task>_<origin>.fcplan**（文本）：FCclasses 任务启动器，按 **fcclasses.plan.json** 逐个执行 ready 作业并生成汇总。

- **fcclasses.plan.json** / **fcclasses.summary.json**（JSON）：FCclasses 的规划记录与执行状态。
- **fcclasses.report.txt** / **fcclasses.report.json** / **fcclasses.report.tsv**（文本/JSON/TSV）：FCclasses 任务级通道状态、速率总和、寿命和量子产率报告。
- **task.manifest.json**（JSON）：任务工件清单，记录输入文件、workflow、主输出文件、依赖、程序参数及命名 **%source** 绑定；来源记录包含声明、解析路径、staged XYZ、电子态与对应 **mod_operations**，供命令刷新和后续工具恢复。
- **.bane.taskmeta.json**（JSON）：结果快照导出时写出的任务元数据 sidecar。
- **.bane.result.user.kv**（文本）：**result** DSL 的用户补充结果缓存，仅在有用户结果写入时出现。
- **.bane.dbcollect.user.kv**（文本）：**dbcollect** / **atomvec** 的 record values 缓存，用于 **--record-kv** 合并。
- **.bane.values.kv**（文本）：运行期细粒度 values 缓存，承载 **atoms.N.*** 等 atom-level 值，供后续任务读取。
- **.bane.result.kv**（文本）：当前任务的轻量级结果快照，在结果导出链路启用时生成。
- **fail/**、**fail1/** 等（目录）：**restart** 时归档已有工件的目录。
- **__banetask_finalize__/** 或自定义 **finalize.task_name** 目录（目录）：启用 **finalize** 后由系统按需物化的隐式收尾任务目录，内部会包含对应的脚本输入、workflow 与命令文件。

5 结果处理与归档

本章说明运行结果的层次、存储位置、规范属性、Derive、Plot、artifact 和 **.btdb** 归档。

5.1 结果模型、快照与数据库

结果链路是 BaneTask 工程化能力的重要组成部分。系统会把日志中的关键信息提炼为快照、记录文件和派生数据库，以便支撑条件判断、批量查询和项目级统计。

5.1.1 集中式事实、缓存与结果文件

- **.bane/cache/snapshots/*.kv**（文本）：集中式轻量快照副本，属于可重建缓存。
- **.bane/store/records/*.json**（JSON）：数据库 ingest 使用的当前规范化记录，属于事实源。
- **.bane/store/revisions/<run_id>/*.json**（JSON）：已有记录再次解析前保存的历史 envelope，属于事实源。
- **.bane/store/objects/sha256/**（二进制/文本）：按内容哈希去重的输入、输出、workflow、结构和波函数对象，属于事实源。
- **.bane/run/db-queue/*.ready**（标记文件）：表示存在新记录待同步的临时 ready 标记。
- **.bane/cache/project.db**（二进制）：可删除并重建的 SQLite 查询缓存。
- **Results/Recipes/**（文本）：实际使用的 banewfn 配方，以及规范化后的 Derive/Plot DSL 视图。
- **Results/Derived/*.derived.json**（JSON）：项目级 **@derive** 生成的标量和表格派生结果。

- **Results/Method/method.md** / **method.json**（文本/JSON）：**btproc method** 或 **finalize.method.generate** 生成的方法学说明与结构化摘要。
- **Results/FCclasses/rate_report.txt** / **rate_report.json** / **rate_report.tsv**（文本/JSON/TSV）：**finalize.fcclasses.report** 生成的 case 或 project 级 FCclasses 速率汇总。
- **<name>.btdb**（ZIP64 单文件）：结项时对 current records、历史 revisions、记录可达对象、Recipes 和可用 **.bt/.projbt** 上下文生成的不可变快照；放置位置由 **--out** 指定。

project.db 由当前 records、历史 revisions 和工件描述符派生生成；长期事实位于 **.bane/store/records/**、**.bane/store/revisions/** 与 **.bane/store/objects/**。**%when** 的条件判断读取任务本地轻量快照，集中副本位于 **.bane/cache/snapshots/**。日常下载和讨论主要面向 **Results/**；彻底结项时可将事实源导出为一个 **.btdb**，而不是保存 SQLite、queue 或 snapshot。相关查询与归档能力通过 **btdb case/project/archive/compare** 完成。

5.1.2 结果链路总览

当任务启用结果快照导出时，系统会在当前任务目录、**.bane/** 存储区和 **Results/** 人工结果视图中写出彼此关联的文件。结果入口是 **btproc result**；默认接入这一结果链路的主要是带输入文件的 Gaussian、GAMESS、ORCA、AMESP、MOPAC 与 BDF 任务，具体取决于当前安装版本支持的日志解析后端；XTB 等任务可通过 **result** / **dbcollect** 或 extxyz 转换显式补充记录。方法学说明属于独立的 **Results/Method/** 输出链路，由 **btproc method** 或 **finalize.method.generate** 生成。

结果提取会从同一次解析中生成轻量 payload、规范 **bane.qc.document.v2** JSON、最终 XYZ 和数据库字段。结果档位 **summary** / **standard** / **full** 控制解析后 payload 的详细程度；数据库工件档位 **none** / **minimal** / **standard** / **full** 则独立控制哪些原始文件进入长期仓库。常规任务建议使用默认结果档位 **standard** 与数据库工件档位 **standard**：前者保留常用结构化结果，后者保留未来重新解析所需的主输入、主输出、workflow、最终结构和可移植文本波函数。

```
任务目录/
  .bane.taskmeta.json
  .bane.result.user.kv
  .bane.dbcollect.user.kv
  .bane.values.kv
  .bane.result.kv

.bane/
  store/
    records/<run_id>.json
    revisions/<run_id>/<parse_revision_id>.json
    objects/sha256/<prefix>/<sha256>
  cache/
    snapshots/<run_id>.kv
    project.db
  run/
    db-queue/<run_id>.ready

Results/
```

```
  Recipes/
    banewfn/<inputname>/<ordinal>-<recipe>.bwc
    derive/<derive-id>.btder
    plot/<plot-id>.btplt
  Method/
    method.md
    method.json
```

5.1.3 .bane.taskmeta.json

.bane.taskmeta.json 由命令生成器在结果快照导出启用时写出，用于保存当前任务的上下文元数据，使后续的结果记录、数据库同步和问题追踪都能知道“这条结果是从哪个任务、哪个输入、哪组头部变量中产生的”。当前包含以下核心字段：

- **record_type**：固定为 **banetask.result_metadata**。
- **schema_version**：当前为 **1**。
- **taskName**：任务名。
- **originName**：原始输入基名。
- **inputName**：**<task>_<origin>**。
- **taskPath**：任务目录绝对路径。
- **strucId**：结构标识；优先来自 **%meta strucid**，否则由来源继承或按新结构自动生成。
- **headerVariables**：YAML 头部元数据与系统注入变量。

结构标识解析规则如下：显式 **%meta strucid** 拥有最高优先级；从 **origin** 或文件源开始的任务通常使用源结构基名；从结构化集合的某一帧开始时会形成包含帧号的结构标识；从上游任务继续的任务会优先继承上游快照中的 **struc_id** / **strucId**。当当前任务会改变几何，例如显式几何修改，或 Gaussian **opt** / **irc** / **scan** 类任务，系统会为该任务生成新的结构标识，通常形如 **<task>_<origin>**。

headerVariables 中常见系统注入键包括：

- **task_name**
- **template_task_name**
- **use_guess**
- **guess_source_task**
- **matrix_variant_key**
- **matrix.***

5.1.4 .bane.result.user.kv

该文件用于暂存 **%process** 或 **%preprocess** 中通过 **result** DSL 写入的用户结果。它的存在使得“程序原生提取结果”和“用户自定义补充结果”能够在同一条结果链路中合并。例如：

```
soc=123.45
note=final structure accepted
```

当存在该文件时，任务收尾阶段的 **btproc result** 会将其并入 **.bane.result.kv** 和 **records/*.json**。

5.1.5 .bane.dbcollect.user.kv

该文件用于暂存 dbcollect DSL 写入的 record values。这些键值会进入 records/*.json 和 SQLite 派生表；.bane.result.user.kv 则面向轻量 .bane.result.kv。因此 .bane.dbcollect.user.kv 适合承载原子级、片段级或其他大批量分析字段。atomvec 也会把派生出的 atom values 追加到该文件，使后续 btproc result 能统一写入 record。

例如：

```
atoms.1.charge=-0.123
atoms.2.charge=0.045
fragments.ring.spin=0.12
```

btdb 同步时会识别 atoms.<index>.<prop> 与 fragments.<id>.<prop> 这类键，并额外展开到原子级与片段级明细表中。

5.1.6 .bane.values.kv

.bane.values.kv 是运行期可读取的细粒度 values 文件，用于承载 atoms.N.* 等 atom-level 字段。它位于轻量 .bane.result.kv 和数据库 record 之间：后续任务可以通过依赖快照读取这些键，轻量结果快照则继续保留 scalar 字段。

例如：

```
atoms.1.charge.resp2=-0.150000
atoms.2.charge.resp2=0.150000
```

依赖加载时，BaneTask 会先读取上游 .bane.result.kv，再叠加 .bane.values.kv。因此后续任务既可以继续使用普通 scalar 快照字段，也可以在需要时读取 atoms.1.charge.resp2 这类运行期 values。

5.1.7 .bane.result.kv

.bane.result.kv 是供 %when 与本地检查直接读取的轻量级快照，也是整个结果链路中最贴近“生成阶段条件判断”的一层。常见字段示例如下：

```
schema_version=1
program=gaussian
energy=-123.456789
scf_energy=-123.400000
free_energy=-123.350000
nimag=1
imag1=-612.23
converged=true
```

这一快照层有三个重要特点：其一，%when 直接读取该轻量快照；其二，当文件缺失或过期时，系统会自动根据日志尝试补建；其三，它还会同步复制到 .bane/cache/snapshots/<run_id>.kv，从而形成集中式快照缓存。依赖加载时如果同目录存在 .bane.values.kv，这些运行期 values 会叠加到快照映射中，供需要 atom-level 字段的后续任务读取。

5.1.8 records/*.json

records/*.json 是数据库层的当前记录。数据库同步与重建读取这些 JSON envelope，并据此形成 case 级或 project 级查询对象。schema v2 记录通常包含：

- run_id、project_key、case_key、task_name 等计算身份；
- parse_revision_id、parsed_at、parser_name、parser_version 等解析身份；
- 通用指标、扁平化 values、来源任务与头部变量；
- artifacts 工件描述符，其中包含 role、逻辑文件名、原始相对路径、SHA-256、字节数、MIME 类型和仓库相对路径；
- source_artifact_sha256 与可选的 reparsed_from_revision_id，用于把当前解析结果追溯到主输出和前一解析 revision。

SQLite 仍是可删除并重建的派生索引；长期证据由 .bane/store/records/、.bane/store/revisions/ 与 .bane/store/objects/ 共同保存。

5.1.9 revisions/<run_id>/*.json

每次对已有 record 再次执行 btproc result 或 btdb ... reparse 前，被替换的 envelope 会以不可变 revision 的形式写入该目录。revision 文件保留当时的 parser 版本、结构化结果和工件引用；当前 record 继续位于 records/<run_id>.json。btdb sync 与 btdb rebuild 会同时导入当前记录和历史 revision，但只有当前记录进入 task_runs / task_flat_latest，历史记录进入 parse_revisions。

5.1.10 .bane/store/objects/sha256/

数据库工件对象采用内容寻址存储。每个对象相对于 .bane/store/ 的路径为 objects/sha256/< 前两位 >/< 完整 SHA-256>；内容相同的输入、输出或波函数只保存一次。record 和 SQLite 只保存描述符，不把大文件内容直接写入数据库列。case 同步先将工件归档到 case 的对象存储；project 同步再把引用到的对象本地化到 project 的对象存储，因此只要保留 <project_root>/.bane/store/，就能够在不依赖原 case 绝对路径的情况下重建 project 查询缓存。若对象已损坏但任务目录中的原文件仍存在，后续 sync 可按描述符哈希重新归档并修复该对象。

工件角色采用稳定语义，例如：

- input.primary / input.auxiliary：主输入与辅助输入；
- output.primary / output.auxiliary：主输出与附加输出；
- manifest.task、workflow.primary、workflow.script：任务 manifest 与实际 workflow；
- structure.final：最终结构；
- wavefunction.portable：.fchk、.fch、.wfn、.wfx、.mwfn、.molden、.molden.input 等可移植文本波函数；
- wavefunction.native：.chk、.gbw、.rwf 等程序原生波函数或 restart 文件；
- result.raw / result.imaginary_modes：解析时的原始结构化 payload 与虚频明细。

数据库工件档位如下：

- none：不新增文件工件，仅保留记录中已有描述符；
- minimal：新生成 record 时归档原始结构化 payload，并保存可发现的 task manifest 与最终结构；

- **standard**：在 **minimal** 基础上保存主输入、主输出、workflow 与可移植文本波函数；这是默认档位；
- **full**：在 **standard** 基础上保存 manifest 声明的附加输出和原生波函数文件。

可通过命令行或环境变量控制策略：

```
btdb project sync --path ./demo_project --artifacts standard
btdb project sync --path ./demo_project --artifacts full
↪  --include-native-wavefunctions
btdb project sync --path ./demo_project --max-artifact-bytes 2147483648

export BANETASK_DB_ARTIFACT_PROFILE=standard
export BANETASK_DB_INCLUDE_NATIVE_WAVEFUNCTIONS=1
export BANETASK_DB_MAX_ARTIFACT_BYTES=2147483648
```

--max-artifact-bytes 0 或未设置环境变量表示不限制单个工件大小；超过非零限制的候选文件会跳过并给出 warning。解析结果档位与数据库工件档位彼此独立，不应把 **btproc result --profile full** 视为主输出长期归档的替代品。

5.1.11 .bane/run/db-queue/*.ready

.bane/run/db-queue/*.ready 用于表示对应 run 存在新的记录可供同步。数据库同步命令会优先读取这一临时队列执行增量 ingest，从而避免每次都全量扫描全部记录。成功 ingest 后 ready 标记会被删除；即使整个 **.bane/run/** 丢失，仍可通过扫描 **.bane/store/records/** 完成全量同步或重建。

5.1.12 project.db

project.db 是最终面向查询与统计的 SQLite 派生数据库。它可以存在于 case 级，也可以存在于 project 级，具体位置如下：

- case 级：**<case_root>/.bane/cache/project.db**
- project 级：**<project_root>/.bane/cache/project.db**

数据库 **schema_version** 为 **5**。**task_runs** 继续保存当前计算记录，并新增 **parse_revision_id**、**parsed_at**、**reparsed_from_revision_id**、**source_artifact_sha256** 与 **artifact_profile**；**parse_revisions** 保存同一 run 的全部解析 revision；**artifact_blobs** 保存内容寻址 blob 的完整性元数据。**task_runs.struc_id** 可用于关联同一结构在不同任务中的计算，原子级与片段级记录仍由 **task_values** 中形如 **atoms.<index>.<prop>**、**fragments.<id>.<prop>** 的键展开。

常用对象包括：

- **task_runs**
- **parse_revisions**
- **artifact_blobs**
- **task_common_metrics**
- **task_values**
- **task_atom_values**
- **task_fragment_values**
- **task_artifacts**

- task_deps
- task_header_vars
- latest_task_runs
- task_flat_latest
- parse_revision_latest
- task_artifact_latest
- task_atom_flat_latest
- task_fragment_flat_latest

5.1.12.1 parse_revisions 与 parse_revision_latest

parse_revisions 用于审计同一计算在不同 parser 版本下的解析历史。parse_revision_latest 仅返回当前 revision；它与 task_flat_latest 中的当前结果一致。

```
SELECT run_id, revision_id, parser_version, parsed_at, is_current
FROM parse_revisions
WHERE run_id = 'opt_case_001'
ORDER BY parsed_at;
```

5.1.12.2 artifact_blobs 与 task_artifact_latest

artifact_blobs 每个 SHA-256 仅保留一行，用于校验仓库 blob；task_artifact_latest 把最新任务记录与工件角色关联起来，适合查找可重新解析的主输出或可移植波函数。

```
SELECT case_key, task_name, role, logical_name, sha256, bytes
FROM task_artifact_latest
WHERE role IN ('output.primary', 'wavefunction.portable');
```

5.1.12.3 task_flat_latest

task_flat_latest 是最常用查询入口。它会对每个 project_key + case_key + task_name 保留最新记录，并把常用字段平铺到一行。

```
SELECT case_key, task_name, energy, nimag
FROM task_flat_latest;
```

5.1.12.4 task_atom_flat_latest

task_atom_flat_latest 会把最新任务记录与 task_atom_values 展开结果关联起来，适合查询 extxyz per-atom 属性、原子电荷、自旋密度或其他按原子编号记录的分析值。字段中的 atom_index 使用 1-based 编号。

```
SELECT case_key, task_name, atom_index, prop_key, value_num
FROM task_atom_flat_latest
WHERE prop_key = 'charge';
```

5.1.12.5 task_fragment_flat_latest

task_fragment_flat_latest 会把最新任务记录与 task_fragment_values 展开结果关联起来，适合查询按片段 ID 记录的能量、布居、偶极或自定义分析值。

```
SELECT case_key, task_name, fragment_id, prop_key, value_num
FROM task_fragment_flat_latest
```

```
WHERE fragment_id = 'ring';
```

5.1.12.6 task_header_vars

task_header_vars 会将 headerVariables 展开成 key/value 形式，便于按 matrix 维度、标签或模板任务名过滤。

```
SELECT case_key, task_name
FROM task_header_vars
WHERE key = 'matrix.method' AND value = 'CAM-B3LYP';
```

5.1.13 .btdb 单文件结项归档

.btdb 的定位是**不可变的导出、迁移和长期保存容器**，不是运行中的 SQLite 数据库，也不用于直接承载 scheduler 或 workflow 状态。在线工作目录继续使用 .bane/store/ 保存事实源；结项时，archive create 对这些事实和用户明确选择的 Results 做可达性裁剪并封装成单文件。恢复后再由 records 与 revisions 重建 .bane/cache/project.db。

归档原则是**描述已经发生的事实，不规定某类任务必须产生什么文件**。BaneTask 不会因为一个 Gaussian、ORCA、IRC、优化、频率或其他已完成任务没有 .fchk、.gbw、.hess、.chk 等文件而拒绝封存。某次计算实际产生并进入 record artifact graph 的内容，以及用户显式选择的 Results 是什么，.btdb 就保存什么。

5.1.13.1 默认封存集合

btdb case/project archive create 会先对目标作用域执行一次全量数据库同步。同步失败或任一 record ingest 发生真实完整性错误时不会发布归档。同步完成后，默认收录：

- .bane/store/records/ 中的 current records；
- .bane/store/revisions/ 中的历史解析 revision；
- 上述 records/revisions 的 storage=sha256 工件描述符实际引用的对象；
- 每个 workspace 下的 Results/Recipes/ 普通文件，包括实际冻结的 .bwc/.inp、.btder、.btplt 和已保存的 settings；
- Results/published.json 以及其中显式选择的普通文件或非空目录内容；
- 归档根与 case workspace 顶层可用的 .bt / .projbt 文件。

默认明确排除：

- .bane/cache/project.db、-wal、-shm；
- .bane/cache/snapshots/；
- .bane/run/、ready marker、lock 和 scheduler 状态；
- record 没有引用的孤儿 SHA-256 对象；
- Results/ 中既不属于 Recipes、也未被 published.json 显式选择的普通文件。

这里不存在后端扩展名白名单或必需文件清单。只有具体事实不一致才会失败，例如 record 引用的对象缺失或 hash 不符、published.json 路径越界，或者用户显式选择的路径在封存时已经消失。

5.1.13.2 发布并显式选择 Results

Process DSL 使用 publish 把文件放入适合日常下载和讨论的 Results/[inputname]/：

```
%process
  publish final.log Cubes/
  publish --pin final-figure.svg summary.csv
```

不带 `--pin` 时只执行 Results 复制；带 `--pin` 时，在复制成功后把实际目标路径登记到：

```
Results/published.json
```

manifest 结构为：

```
{
  "format": "banetask.results.published",
  "format_version": 1,
  "paths": [
    "task_input/final-figure.svg",
    "task_input/summary.csv"
  ]
}
```

路径始终相对于 `Results/`。绝对路径、`..` 穿越、符号链接、空目录和 `published.json` 自身选择会被拒绝。`publish --pin` 只登记本次源路径实际匹配并复制的文件，不会重新扫描目标目录来吸收目录中已有但未由本次命令复制的文件。直接选择非空目录时，`archive create` 递归收录其中普通文件；重叠选择会去重。`%process archive ...` 是 `publish ...` 的别名；规范命令名为 `publish`。

5.1.13.3 容器布局与完整性

第一版格式使用 ZIP64 的 store 模式，扩展名为 `.btdb`。容器可被通用 ZIP 工具列出和抢救，但格式语义由根目录的 `btdb.json` 定义：

```
study.btdb
├── btdb.json
├── records/
│   ├── current/<workspace-id>/...
│   └── revisions/<workspace-id>/...
├── objects/sha256/< 前两位 >/< 完整 SHA-256>
├── results/recipes/<workspace-id>/...
├── results/published-manifests/<workspace-id>/published.json
├── results/published/<workspace-id>/...
└── workflow/<workspace-id>/...
```

`btdb.json` 记录：

- 格式版本、scope、创建时间、producer 版本和 artifact profile；
- project root、case 等 workspace 的稳定 ID、角色和归档内相对路径；
- 每个 payload 的 kind、归档路径、恢复路径、字节数和 SHA-256；
- current record、revision、对象、Recipes、显式发布结果、workflow context 的统计；
- 由规范化 workspace 与 entry 清单计算的 `content_root_sha256`；
- 由 content root 前缀形成的 `archive_id`。

相同内容的对象在 project 归档中只写入一次；manifest 可以为该对象登记多个恢复路径，使其恢复到各个 case 的 `.bane/store/objects/sha256/`。ZIP 的 CRC 用于容器层快速检测，

SHA-256、record 引用检查、published selection 检查和 content root 用于语义层完整性验证。

SHA-256 只能检测内容是否被修改或损坏，不能证明归档是谁创建的。当前格式没有数字签名、加密或密码保护。

5.1.13.4 创建归档

```
# 单 case
btdb case archive create \
  --path ./caseA \
  --out caseA.btdb

# 整个 project
btdb project archive create \
  --path ./demo_project \
  --out demo_project.btdb
```

创建入口接受与 **sync** 相同的工件发现策略。策略只控制**发现并收录实际存在的文件**，不会把未产生的扩展名变成要求：

```
btdb project archive create \
  --path ./demo_project \
  --out demo_project.btdb \
  --artifacts standard \
  --max-artifact-bytes 2147483648

btdb project archive create \
  --path ./demo_project \
  --out demo_project-full.btdb \
  --artifacts full \
  --include-native-wavefunctions
```

其他选项：

- **--overwrite**：替换已有普通 **.btdb**；未指定时拒绝覆盖。实现先完整生成并验证临时归档，再提交替换，因此创建或验证失败不会截断旧文件。
- **--no-recipes**：不收录 **Results/Recipes/**。通常不建议，因为这些文件体积小且保存实际分析方法。
- **--no-published-results**：不收录 **Results/published.json** 及其选择的普通 Results。
- **--no-workflow-context**：不收录顶层 **.bt/.projbt**。通常仅在明确只需要数据库事实时使用。

输出文件不能位于目标作用域的 **.bane/store/**、**.bane/cache/** 或 **.bane/run/** 内；归档源中的 record、对象、Recipe、显式发布结果和 workflow context 必须是普通文件，符号链接会被拒绝，以免无意中把 workspace 外部内容带入归档。输出 **.btdb** 及其同名临时文件不会作为 Recipe 或其他 payload 收入自身；若 **published.json** 明确选择了输出文件，创建会失败并要求调整输出位置或选择清单。

创建流程为：

1. 执行全量 **sync**。
2. 收集 current records 与 revisions。

3. 校验并规范化 artifact 描述符。
4. 标记记录可达对象并去除孤儿对象。
5. 收集 Recipes、published Results 与任务上下文。
6. 写入 **<out>.tmp-<unique>**。
7. 完整执行 **archive verify**。
8. 发布最终 **.btdb**。

5.1.13.5 查看与校验

```
btdb archive info demo_project.btdb
btdb archive verify demo_project.btdb
```

info 只读取并验证 **btdb.json** 的结构与语义，不提取 payload；用于查看 archive ID、scope、producer、记录数、对象数、Recipes 数、published Results 数、字节数和总文件大小。

verify 会完整读取归档并检查：

- ZIP64 目录、entry 数量、CRC、大小和不支持的压缩/加密/multi-disk 标志；
- manifest 的格式、scope、workspace、entry kind、统计和 content root；
- 归档路径与恢复路径是否安全、唯一且符合 kind 对应布局；
- 每个 payload 的字节数和 SHA-256；
- current/revision record 是否为有效 JSON，**record_path** 是否与恢复位置一致；
- 每个 SHA-256 artifact 描述符是否引用归档内对象，路径、大小和 hash 是否一致；
- **Results/published.json** 是否有效，每个 published payload 是否被清单显式选择，并且每个清单选择是否至少对应一个已归档普通文件；
- 是否存在未被任何 record/revision 引用的孤儿对象。

从集群下载后，应在本地再次执行 **btdb archive verify** 并确认返回码为 0，再决定是否删除集群工作目录。

5.1.13.6 恢复或导入

```
btdb archive restore demo_project.btdb \
  --out ./restored_project

# import 当前是 restore 的语义别名，不会合并到已有项目
btdb archive import demo_project.btdb \
  --out ./imported_project
```

默认恢复行为：

1. 在同一个已打开的归档实例上执行完整校验和恢复；每个 payload 写入临时目标时再次核对 manifest 字节数和 SHA-256，校验通过后才提交目标文件；
2. 按 manifest 恢复 **.bane/store/records**、**revisions**、对象、**Results/Recipes**、**Results/published.json**、显式发布结果和可用 **.bt/.projbt**；
3. 在 **.bane/archive/btdb.json** 保存本次恢复使用的 manifest；
4. 必要时补一个仅用于 case/project 发现的最小 **.bt/.projbt** 占位文件；
5. 从恢复后的事实源重建 **.bane/cache/project.db**。

选项：

- **--no-rebuild**：只恢复事实源，不生成 SQLite；适合仅恢复文件并稍后重建 SQLite。

- `--materialize`：在重建索引后，把已登记工件额外恢复到普通的 `Recovered/<case>/<task>/<run>/...` 目录。归档没有 artifact 时该操作正常成功，恢复数量为 0。该选项不能与 `--no-rebuild` 同时使用。
- `--overwrite`：允许向非空目录恢复并替换同名普通文件。恢复器拒绝符号链接目标和目录穿越。

恢复按单个文件原子写入，但整个目录恢复目前不是事务性的；磁盘写入失败可能留下部分目录。此时应删除不完整目标目录后重新执行。`import` 目前不会处理已有项目中的 run ID 冲突、current revision 选择或对象合并。

5.1.13.7 当前归档边界

当前 `.btdb` 已能在删除原 case/project 后恢复 records、revision、对象、Recipes、显式发布 Results 和 SQLite 查询能力，但仍有以下边界：

- Recipes 作为实际文件被保存，但尚未成为带输入、输出和工具版本边的第一等 provenance 节点。
- 尚未生成能够独立执行所有恢复 `.btder/.btplt` 的规范化 effective `.bt`；归档只保存找到的顶层任务上下文。
- `published.json` 当前是路径选择清单，尚未记录成果标题、角色、生成 recipe、输入对象和输出对象之间的完整 provenance 图。
- 尚无数字签名、加密、增量归档链或已有项目合并导入。
- 第一版使用无压缩 store 模式；重点是可校验性、通用 ZIP 可读性和恢复可靠性，而不是最小容器体积。

5.2 规范结果查询

统一量化结果查询用于在 `btproc result`、Plot DSL 和 Derive DSL 中用同一组属性名读取任务结果。这样，同一个能量、频率、优化轨迹或激发态字段在命令行、绘图和派生结果中都使用一致的名称、单位和缺失状态。

5.2.1 属性名规则

规范属性名大小写不敏感，首尾空白会被忽略。别名只用于便捷输入；归档、日志和来源信息使用规范名。

属性列表以统一规范属性目录为准。命令行属性发现、属性说明、直接查询、Plot DSL 和 Derive DSL 读取同一套规范目录；BaneTask 仅维护 `energy`、`freq`、`osc`、`dipole`、`soc` 等面向 DSL 的便捷别名。

`result(task, property)` 会使用规范属性目录推导任务级解析需求，并按属性的 shape、axis、dimension、unit 和缺失状态输出结果。

常用写法：

```
value energy(ts)
y result(self, "qc.optimization.max_force")
```

轻量 `.bane.result.kv` 字段仍通过 `kv(...)` 访问，不会自动混入规范属性名。

5.2.2 标量属性

规范属性	维度/单位	常用别名
`qc.energy.primary`	energy / hartree	`energy`, `primary_energy`
`qc.energy.scf`	energy / hartree	`scf_energy`
`qc.energy.correlated`	energy / hartree	`correlated_energy`, `post_scf_energy`, `post_hf_energy`, `posthf_energy`
`qc.energy.double_hybrid`	energy / hartree	`double_hybrid_energy`, `doublehybrid_energy`, `dh_energy`
`qc.energy.composite`	energy / hartree	`composite_energy`, `composite`, `cbs_qb3_energy`, `cbsqb3_energy`
`qc.energy.composite_zero_k`	energy / hartree	`composite_zero_k_energy`, `composite_0k_energy`, `cbs_qb3_0k_energy`, `cbsqb3_0k_energy`
`qc.thermochemistry.gibbs_energy`	energy / hartree	`free_energy`, `gibbs_energy`
`qc.thermochemistry.gibbs_correction`	energy / hartree	`gibbs_corr`, `gibbs_correction`
`qc.thermochemistry.zero_point_energy`	energy / hartree	`zpe`, `zero_point_energy`
`qc.thermochemistry.thermal_energy`	energy / hartree	`thermal_energy`
`qc.thermochemistry.thermal_correction`	energy / hartree	`thermal_corr`, `thermal_correction`
`qc.thermochemistry.enthalpy`	energy / hartree	`enthalpy`
`qc.thermochemistry.enthalpy_correction`	energy / hartree	`enthalpy_corr`, `enthalpy_correction`
`qc.thermochemistry.entropy_correction`	energy / hartree	`entropy_corr`, `entropy_correction`
`qc.thermochemistry.entropy`	entropy / J/mol/K	`entropy`, `thermo.entropy`
`qc.thermochemistry.cv`	heat capacity / J/mol/K	`cv`, `thermo.cv`, `constant_volume_heat_capacity`
`qc.thermochemistry.cp`	heat capacity / J/mol/K	`cp`, `thermo.cp`, `constant_pressure_heat_capacity`
`qc.thermochemistry.temperature`	temperature / K	`temperature`, `thermo.temperature`
`qc.thermochemistry.pressure`	pressure / atm	`pressure`, `thermo.pressure`
`qc.vibration.imaginary_count`	dimensionless	`nimag`, `imag_freq`, `imaginary_frequency_count`

规范属性	维度/单位	常用别名
qc.hessian.dimension	dimensionless	hessian.dimension, cartesian_hessian.dimension, qc.hessian.dim

5.2.3 序列属性

规范属性	轴	维度/单位
qc.optimization.step	optimization step	step
qc.optimization.energy	optimization step	energy / hartree
qc.optimization.max_force	optimization step	force
qc.optimization.rms_force	optimization step	force
qc.optimization.max_displacement	optimization step	length
qc.optimization.rms_displacement	optimization step	length
qc.hessian.cartesian	hessian element	force constant / hartree/bohr^2
qc.vibration.frequency	vibrational mode	frequency / cm-1
qc.vibration.reduced_mass	vibrational mode	mass / amu
qc.vibration.force_constant	vibrational mode	force constant / mDyne/angstrom
qc.vibration.ir_intensity	vibrational mode	spectral intensity / km/mol
qc.vibration.raman_activity	vibrational mode	Raman activity / angstrom^4/amu
qc.vibronic.transition.energy	vibronic transition	frequency / cm-1
qc.vibronic.transition.intensity	vibronic transition	spectral intensity
qc.vibronic.spectrum.energy	vibronic spectrum point	frequency / cm-1
qc.vibronic.spectrum.intensity_0k	vibronic spectrum point	spectral intensity
qc.vibronic.spectrum.intensity	vibronic spectrum point	spectral intensity
qc.vibronic.duschinsky.row	Duschinsky element	dimensionless, 1-based row index
qc.vibronic.duschinsky.column	Duschinsky element	dimensionless, 1-based column index

规范属性	轴	维度/单位
`qc.vibronic.duschinsky.matrix`	Duschinsky element	dimensionless
`qc.vibronic.shift_vector`	vibrational mode	Gaussian shift-vector value
`qc.vibronic.huang_rhys`	vibrational mode	dimensionless
`qc.spin_orbit_coupling.singlet_index`	spin-orbit coupling element	step/index
`qc.spin_orbit_coupling.triplet_index`	spin-orbit coupling element	step/index
`qc.spin_orbit_coupling.magnitude`	spin-orbit coupling element	energy / cm-1
`qc.excited_state.index`	excited state	step/index
`qc.excited_state.excitation_energy`	excited state	energy / hartree
`qc.excited_state.wavelength`	excited state	length / nm
`qc.excited_state.oscillator_strength`	excited state	dimensionless
`qc.excited_state.total_energy`	excited state	energy / hartree
`qc.excited_state.multiplicity`	excited state	dimensionless
`qc.excited_state.spin_squared`	excited state	dimensionless
`qc.excited_state.singles_character`	excited state	dimensionless / %
`qc.excited_state.major_contribution`	excited state	text
`qc.orbital.energy`	orbital	energy / hartree
`qc.orbital.occupation`	orbital	dimensionless
`qc.orbital.spin`	orbital	dimensionless/text
`qc.orbital.alpha.energy`	orbital	energy / hartree
`qc.orbital.alpha.occupation`	orbital	dimensionless
`qc.orbital.beta.energy`	orbital	energy / hartree
`qc.orbital.beta.occupation`	orbital	dimensionless

三个旋轨耦合属性共享 `(triplet_index,singlet_index)` key，因此可以直接作

为 heatmap 的 x、y、z。`qc.spin_orbit_coupling.magnitude` 的常用别名包括 `soc.magnitude` 与 `soc.matrix`。同一态对同时存在 ORCA `M_S` 与 Cartesian XYZ 分量时，统一查询默认优先 `M_S`，缺失时回退 XYZ；同一分量基组内重复出现的态对保留最后一个矩阵元。

完整属性和别名以运行时属性列表为准：

```
btproc result list-properties
btproc result list-properties --json
```

5.2.4 查看属性说明

```
btproc result describe-property qc.optimization.energy
btproc result describe-property frequency --json
```

文本输出示例：

```
name: qc.optimization.energy
shape: series
dimension: energy
unit: hartree
axis: optimization_step
requirements: energy_history
aliases: step.energy opt.energy
description: Preferred energy at each calculation step.
```

5.2.5 直接查询量化输出

```
btproc result query-property scan.log qc.energy.primary
btproc result query-property scan.log qc.energy.primary --unit kcal/mol
btproc result query-property scan.log qc.optimization.energy --unit kcal/mol
↪  --json
```

标量文本输出为 `value<TAB>unit`；序列文本输出为 `key/value/unit` 表。JSON 输出包含 shape、dimension、unit、axis、规范 property 和实际 selection。`qc.hessian.cartesian` 的 JSON `axis` 为 `hessian_element`，序列键采用 `row,column` 形式表示矩阵元素。Gaussian Duschinsky 矩阵同样使用 `row,column` 键；`qc.vibronic.duschinsky.row`、`qc.vibronic.duschinsky.column` 与 `qc.vibronic.duschinsky.matrix` 三个序列按同一元素轴严格对齐，可直接送入 Plot DSL 的 heatmap。

退出码约定：

- `0`：查询成功；
- `1`：参数、属性名或单位错误；
- `2`：属性合法，但结果不可用。

5.2.6 在 Plot DSL 中使用

Plot DSL 可以直接读取规范属性：

```
@plot curve spectrum
  from task td
  x result(td, "qc.excited_state.wavelength") unit nm
  y result(td, "oscillator_strength")
@end
```

`result(task, property)` 会根据属性说明自动请求所需结果。例如读取激发态波长和振子强度时，会使用激发态结果；读取优化曲线时，会使用优化历史。

5.2.7 在 Derive DSL 中使用

Derive DSL 的标量和表流水线也使用同一组属性名：

```
@derive activation
  let G_r = single_point(sp_R) + gibbs_corr(freq_R)
  let G_ts = single_point(sp_TS) + gibbs_corr(freq_TS)
  emit delta_g = convert(G_ts - G_r, "kcal/mol")
@end
```

无法直接通过快捷函数表达的属性，可使用 `result(task, property)`：

```
let e = result(sp, "qc.energy.primary")
```

5.2.8 结果状态

查询结果会区分不同的不可用状态，便于定位是输入缺失、未解析还是属性不适用。

状态	含义
`present`	已得到类型化结果
`not_requested`	本次规范化没有请求所需分析
`not_parsed`	当前文档档位没有解析该分析
`not_present`	已请求且已解析，但源输出没有该值
`not_applicable`	属性或当前结果类型不适用
`ambiguous`	多个候选值无法安全选取唯一结果
`parse_failed`	解析失败

单位转换会保留属性的轴和来源信息。能量、长度和角度支持跨单位换算；其他已注册维度保留单位元数据，并按原单位输出。不兼容的物理维度会报错。

5.3 `Results/Recipes` 分析配方快照

`Results/Recipes/` 保存本次分析实际采用的配方视图，避免只记录一个模板名称而无法确认历史执行内容。分为三类：

```
Results/Recipes/
  banewfn/<inputname>/
    001-<recipe>.bwc                # 也可能保留原始 .inp 后缀
    001-<recipe>-multiwfn-settings.ini    # Bash workflow
  derive/<derive-id>.btder
  plot/<plot-id>.btplt
```

banewfn workflow 运行时先按既有顺序查找当前目录和 **BANETASK_BANEWFN_PATH** 中的 **.bwc** / **.inp**，再把找到的文件复制到上述目录，并直接使用该副本执行。因此 recipe 文件是实际被 **banewfn** 读取的字节副本，而不是 finalize 时根据模板名事后复制的版本。Bash workflow 还会为每次调用复制 **${MULTIWFN_PATH}/settings.ini**，并通过 **-set** 使用该副本；调用序号用于防止同一任务内多次 **banewfn** 相互覆盖。Windows workflow 当前保存并执行 **.bwc** / **.inp** 副本，但尚未自动冻结 Multiwfn settings。

Derive 和 Plot pipeline 在进入实际求值或渲染前，会把解析并规范化后的声明序列化为 **.btder** / **.btplt**。同一 id 再次执行时会原子覆盖当前 recipe 视图；其结果数据仍分别位于 **Results/Derived/** 和 plot artifact 目录。这里保存的是面向检查和复现的 DSL，不替代 **.derived.json**、绘图数据、renderer 脚本或最终图片。

Results/Recipes/ 当前属于面向人的 Results 视图，并未自动作为独立 artifact 关联到 record。单独复制 **.btder** / **.btplt** 到其他位置执行时，仍必须遵守独立 Derive 脚本 或独立 Plot 脚本 所述的同目录唯一 **.bt** 配对规则。

5.4 Derive DSL

BaneTask Derive DSL 用于在 **.bt** / **.projbt** 中声明项目级结果计算。一个 **@derive** 块读取已有任务结果，执行带 shape、axis、dimension、unit 和 provenance 的表达式或表流水线，并写出可追踪的派生结果。需要给人工快速阅读的最终结论时，也可以在同一个 derive 中声明 report 行并把文本报告写到 case 根目录：

```
Results/Derived/<derive-id>.derived.json
Results/Derived/<derive-id>/<artifact>
Report_<derive-id>.txt              # 仅在 emit report to root 时生成，可自定义文件名
```

当前输出 schema 为：

```
banetask.derived.v1
```

5.4.1 能力总览

适合放入 **@derive** 的工作包括：

- 计算反应能、活化自由能、热化学组合和单位转换；
- 对规范 QM 属性进行标量或序列读取；
- 从一组任务中生成结果表；
- 读取已有 derived document 的标量或表，构成 derive-to-derive 数据流；
- 用 **group by**、**aggregate** / **reduce**、**sort**、**limit** 完成表级汇总；
- 用 **relative**、**rank**、**dense_rank**、**normalize** 完成跨行表计算；
- 从多个任务读取逐原子属性，并按 **atom.id** 严格连接；

- 输出 `.derived.json`、CSV / TSV / JSONL 表文件、标量 `.kv`、atom vector、逐原子 `.kv`、`.extxyz` / `.xyz` / `.chg` artifact，以及面向人工阅读的根目录 report；
- 通过输入 fingerprint 跳过未变化的派生结果。

涉及外部命令、文件复制、输入修改、`extra.yaml` 生成、record / SQLite / snapshot 写入或任意外部目录写入等操作时，继续使用 `%preprocess`、`%process` 或专用命令行工具。`@derive` 的普通 artifact 输出被限制在 `Results/Derived/<derive-id>/` 下；`emit report to root [file]` 是唯一的根目录文本报告出口，且 file 只能是文件名。纯数值结果整理优先写在 `@derive` 中。

5.4.2 声明位置与基本结构

`@derive` 是 `.bt` / `.projbt` 顶层声明，不属于某个 `$task` 块：

```
@derive activation
  let G_r = single_point(sp_R) + gibbs_corr(freq_R)
  let G_ts = single_point(sp_TS) + gibbs_corr(freq_TS)

  require nimag(freq_R) == 0
  require nimag(freq_TS) == 1

  emit delta_g = convert(G_ts - G_r, "kcal/mol")
@end
```

名称同时作为 derive id。名称必须是标识符，允许字母、数字、下划线、点和连字符，首字符必须是字母或下划线：

```
activation
conformer.ranking
resp2-charges
```

不允许嵌套 `@derive`。每个块至少包含一个 sink：标量 `emit`、标量文件 `emit value ... to kv`、根目录报告 `emit report to root [file]`、`emit table`、表文件 `emit table ... to csv|tsv|jsonl`、`emit atomvec`、`emit kv` 或 `emit artifact`。`report` 行本身会进入 JSON，但不是 sink；若希望生成文本报告，须再写 `emit report`。

Derive 以逻辑语句解析。只要圆括号尚未闭合，或物理行以逗号、`+`、`-`、`*`、`/` 等连续运算符结束，同一语句可以跨多行；函数调用和列表允许在右括号前保留尾随逗号。字符串和 `{{ task.key }}` snapshot 引用不能跨物理行。`#` 可用于行尾注释；引号和函数括号内的 `#` 不作为注释起点。缩进本身不触发普通表达式续行，只用于 `emit ...:` 等显式子块。

顶层 `define:` 变量会在 Derive 解析阶段应用，因此任务名、选择器和表达式中可继续使用项目已有的变量展开规则。

5.4.3 参数复用与公式声明

`profile`、`formula` 和 `bind` 用于减少重复声明。它们不能作为值赋给变量或直接输出；`Results/Recipes/derive/<derive-id>.btder` 中的规范化 recipe 会写出完整的函数参数、参数替换结果和 `bind` 生成的 `let` 语句，便于检查方法参数、任务引用和派生依赖。

5.4.3.1 参数 profile

profile 保存某个函数的一组不可变具名参数，适合统一重复的热力学模型参数：

```
profile thermo_std for thermo = (
  model="[model]",
  temperature=[temperature],
  pressure=[pressure],
  zpe_scale=[zpe_scale],
  enthalpy_scale=[enthalpy_scale],
  entropy_scale=[entropy_scale],
)

let G_R = thermo(
  freq_R,
  using=thermo_std,
  quantity=gibbs_free_energy,
  electronic_energy=E_R,
)
```

profile 条目必须全部使用具名参数，不能继承另一个 profile。**using=** 只能用于 profile 所声明的目标函数；调用点与 profile 重复声明同一参数时直接报错，不采用静默覆盖。**thermo** profile 不允许隐藏 **task**、**quantity** 或 **electronic_energy**，这些调用特异参数必须留在调用点。profile 只在当前 **@derive** 内有效，并且必须先定义后使用。

5.4.3.2 纯表达式 formula

formula 定义可重复使用的单表达式公式。调用时按参数替换表达式，不生成可传递或输出的函数值：

```
formula activation(ts, reactant) = ts - reactant
let dG = activation(G_TS, G_R)
```

它也可组合 profile，但所有任务引用和选择器都必须通过参数或字符串字面量显式传入：

```
profile thermo_std for thermo = (
  model=rrho,
  temperature=298.15,
  pressure=1.0,
)

formula state_gibbs(freq_task, energy_task) = thermo(
  freq_task,
  using=thermo_std,
  quantity="gibbs_free_energy",
  electronic_energy=single_point(energy_task),
)
```

formula 体只能包含一个纯表达式；禁止递归、相互递归和 snapshot 引用。除参数及 **using=** 中已声明的 profile 名外，公式中的其他名称必须写成字符串字面量或通过参数传入。内建函数名、Derive 关键字和统一结果属性名不能作为 formula 名。formula 必须在调用前定义；单次展开的最大嵌套深度为 64。

5.4.3.3 `bind thermo(...) to (...)`

一次声明多个热力学 quantity 时，可用 projection binding：

```
bind thermo(
  freq_R,
  using=thermo_std,
  electronic_energy=E_R,
) to (
  ZPE_R = zero_point_energy,
  H_R = enthalpy,
  S_R = entropy,
  G_R = gibbs_free_energy,
)
```

当前 `bind` 只接受直接的 `thermo(...)` 调用，并展开为多条普通 `let`；基础调用中不能再写 `quantity=`。projection 左侧必须是简单标识符，右侧必须是 thermo quantity 标识符。

5.4.4 执行时机和命令行

5.4.4.1 默认完整模式

普通执行：

```
banetask case.bt
```

通常按以下阶段处理：

```
任务生成/同步
未提前执行的 Derive pipeline
Plot pipeline
finalize
```

引用 `{{ derived(...) }}` 的任务在准备前会先保证对应 derive 结果可用，因此该 derive 及其依赖可以早于任务后的 Derive pipeline 执行。未被任务占位符引用的 derive 在任务生成/同步完成后统一处理。

普通完整模式中，未被任务占位符引用的 derive 若缺少任务结果，会被跳过，便于首次生成 workflow 时保留派生声明。任务占位符所需的 derive 采用严格处理：缺少输入结果会使引用任务失败。计算完成后再次执行即可生成其余派生结果。同一 `.bt` / `.projbt` 中的 `@plot from derived(...)` 可以读取本轮写出的 `.derived.json`。隐式 finalize 在 derive 与 plot 完成后再物化，因此可面向本轮派生表、图像和 artifact 执行收尾动作。`--plots-only` 不会先执行 derive；此时上游 derived 结果必须已经存在。

5.4.4.2 仅执行 derive

```
banetask case.bt --derives-only
```

该模式不准备任务，只读取已有结果并执行 derive。缺失任务结果按严格错误处理，适合计算完成后的正式结果生成。

只执行一个声明：

```
banetask case.bt --derives-only --derive activation
```

5.4.4.3 查看依赖和解析需求

```
banetask case.bt --explain-derives
```

只查看一个声明：

```
banetask case.bt --explain-derives --derive resp2
```

--explain-derives 不求值和写文件，只显示 derive id、引用任务、derive 依赖、每个任务需要读取的结果类别和轻量 snapshot 依赖。

5.4.4.4 强制重新执行

默认执行会读取已有 **.derived.json** 的 **provenance.input_fingerprint**。当 fingerprint 一致，且 JSON 记录的 artifact 文件仍存在并且 fingerprint 匹配时，derive 会被判定为 up to date 并跳过。

强制重新执行：

```
banetask case.bt --rerender-derives
```

--rerender-derives 可与 **--derive <name>** 联合使用。

5.4.4.5 禁用 derive

```
banetask case.bt --no-derives
```

--no-derives 不能与 **--derives-only**、**--explain-derives**、**--rerender-derives** 或 **--derive** 同时使用。

5.4.5 语句顺序

一个 derive 由标量前置区和可选表流水线组成：

```
let / require / assert / emit scalar / report / emit value to kv / emit report
 ↪  to root
from
join
where
select
group by
aggregate / reduce
sort
limit
emit table / emit table to csv|tsv|jsonl / emit atomvec / emit kv / emit
 ↪  artifact
```

当前顺序约束如下：

1. **let**、**require**、**assert**、标量 **emit**、**report**、标量文件 **emit value ... to kv** 和 **emit report to root [file]** 必须出现在 **from** 之前；

2. 每个 derive 最多一个 `from`；
3. `join` 只允许出现在 atom 表中，并且必须位于 `where` / `select` / `group by` / `sort` / `limit` 之前；
4. 所有 `where` 必须位于第一个 `select` 之前；
5. `select` 必须位于 `group by` 之前；
6. `group by` 最多一个，并且必须位于 `aggregate` / `reduce`、`sort` 和 `limit` 之前；
7. `aggregate` / `reduce` 必须位于 `group by` 之后、`sort` 和 `limit` 之前；
8. `sort` 必须位于 `limit` 之前；
9. 表流水线必须至少包含一个 sink；
10. 多个 sink 可以共享同一条表流水线。

例如：

```
@derive mixed_output
  let reference = energy(ref)
  emit reference_energy = convert(reference, "kcal/mol")

  from tasks("conf_*")
  where present(energy())
  select task.name
  select E = convert(energy(), "kcal/mol")
  sort E asc
  limit 10
  emit table conformers
@end
```

5.4.6 标量语句

5.4.6.1 `let`

保存局部派生变量：

```
let G_r = single_point(sp_R) + gibbs_corr(freq_R)
let G_ts = single_point(sp_TS) + gibbs_corr(freq_TS)
```

局部变量只在当前 derive 中可见。变量保存完整的类型、物理维度、单位、序列轴和来源信息。

5.4.6.2 `require` / `assert`

两者当前语义相同，用于在输出前验证科学条件：

```
require nimag(freq_R) == 0
assert nimag(freq_TS) == 1
```

支持的条件形式：

```
truthy
==
!=
<
<=
```

```
>
>=
and or not
in
matches
```

数值比较会检查物理维度，并在相同维度下把右侧换算到左侧单位。`==` / `!=` 使用相对于数值量级的浮点容差。文本比较、成员判断、通配匹配和布尔组合的规则与表流水线中的 `where` 一致。

`require` 失败会使整个 derive 失败。可用 `else` 提供面向方法学检查的自定义诊断：

```
require [temperature] > 0 else "temperature must be positive"
require nimag(freq_TS) == 1 else "transition state must contain exactly one
 ↪  imaginary mode"
```

自定义消息会经过顶层变量替换；条件本身仍按原有类型和单位规则求值。

5.4.6.3 标量 `emit`

```
emit delta_e = convert(energy(product) - energy(reactant), "kcal/mol")
emit max_force = max(result(opt, "qc.optimization.max_force"))
```

输出会进入 JSON 的 `values` 数组，保存名称、规范化表达式、shape、数值或序列、物理维度、单位、属性来源和选择来源。

已有变量可省略重复表达式；`unit` 后缀在写入前执行单位换算：

```
emit delta_e
emit delta_e_kcal = delta_e unit "kcal/mol"
emit activation_gibbs_energy unit "kJ/mol"
```

`emit name` 等价于 `emit name = name`。带 `unit` 的 emit 只接受可转换的已有量纲，不能用来给无量纲常数附加物理维度；构造物理量应使用 `quantity()`。

同一结果需要同时输出和报告时，可用一层修饰块：

```
emit activation_gibbs_energy = G_TS - G_R:
  unit "kcal/mol"
  report "Activation Gibbs energy"
```

该形式先求值一次，再执行单位换算并写入 emitted value；由修饰块产生的 `report` 读取同一个已转换结果，不会重新计算右侧表达式。`unit` 最多声明一次，`report` 可声明多行。根目录报告仍需显式写 `emit report to root "file.txt"`。

5.4.6.4 标量文件 sink `emit value`

标量文件 sink 用于把当前 derive 标量区已经产生的变量写成 `.kv` artifact：

```
emit delta_g = convert(G_ts - G_r, "kcal/mol")
emit value delta_g to kv "activation.values.kv"
```

`emit value` 只支持 `kv` 格式。被引用的名称必须来自当前 derive 中前面的 `let` 或标量 `emit`。输出路径必须是相对路径，不能包含 `..`，实际文件写入：

```
Results/Derived/<derive-id>/<path>
```

该 artifact 会同时记录在 `.derived.json` 的 `artifacts` 数组中，kind 为 `scalar-kv`。

5.4.6.5 报告行与根目录报告 `report` / `emit report`

`report` 用于定义面向人工阅读的结论行；`emit report` 用于把这些结论行渲染成 case 根目录下的文本报告：

```
@derive pka_summary
  emit G_acid_aq = -123456.789012
  emit G_anion_aq = -123123.456789
  emit G_Hplus_aq = -270.29
  emit DG_deprot = 6.123456
  emit pka = 4.489

  report "Final aqueous free energy of acid AH" = G_acid_aq
  report "Final aqueous free energy of anion A-" = G_anion_aq
  report "Aqueous proton free energy" = G_Hplus_aq
  report "Delta G deprotonation" = DG_deprot
  report "pKa" = pka
  emit report to root "Report_pka_summary.txt"
@end
```

语法规则：

```
report <label> = <scalar-expression>
emit report to root
emit report to root "<file-name>"
```

`report` 与 `emit report` 必须位于 `from` 之前。`report` 的右侧表达式必须能求得标量；可以直接写表达式，也可以引用前文 `let` 或标量 `emit` 产生的变量。标签可写成标识符或字符串，包含空格时必须加引号。

`emit report` 当前只接受 `to root [file]`。省略 path 时，默认文件名为：

```
Report_<derive-id>.txt
```

根目录报告路径必须是单个相对文件名，不能包含目录、绝对路径或 `..`。实际写入位置是当前 case / project 根目录，而不是 `Results/Derived/<derive-id>/`。

报告文本格式固定为：

```
BaneTask derive report
derive: <derive-id>
status: ok
input fingerprint: <fingerprint>

<label>    = <value> [unit]
...

Canonical result:
  Results/Derived/<derive-id>.derived.json
```

带单位的数值按 6 位小数输出并追加单位；无单位的 dimensionless 数值会去掉多余尾随零。`report` 行会写入 `.derived.json` 的 `report_lines` 数组，`emit report` 会写入 `report_sinks` 数组，并在 `artifacts` 中登记 kind 为 `derive-report`、scope 为 `case-root` 的文本 artifact。

如果已有 `.derived.json` 的 input fingerprint 未变化，但记录的 report artifact 被删除或 fingerprint 不一致，derive 不会简单跳过，而会重新写出缺失或不一致的报告 artifact。

5.4.7 表流水线

5.4.7.1 任务表 `from tasks(...)`

最简单的任务表：

```
@derive inventory
  from tasks("*")
  select task.name
  select task.program
  emit table tasks
@end
```

`tasks()` 支持 shell 风格 pattern：

```
from tasks("conf_*")
from tasks("reactant_*", "product_*")
from tasks(pattern="scan_*", program="gaussian")
from tasks(name="sp_*", template="single_point")
```

支持的选择器：

参数	含义
位置参数	task name pattern，可重复
`pattern=` / `name=`	task name pattern，可重复
`program=`	程序类型 pattern，最多一个
`template=` / `template_task=`	模板任务名 pattern，最多一个

没有 name pattern 时默认使用 `"*"`。如果没有任何任务匹配，derive 失败，不生成空表。

任务表行内可用的文本字段：

字段	含义
`task.name`	展开后的真实任务名
`task.program`	规范程序类型名
`task.template`	matrix / foreach 展开前的模板任务名；不存在时该字段缺失
`task.struc_id`	结构 ID；不存在时该字段缺失
`matrix.<name>`	matrix 轴的原始文本值

数值型 matrix 值也会记录为数值字段。在 `select`、`where`、`sort` 这类可同时接受文本的上下文中，bare `matrix.<name>` 优先按文本读取；需要数值读取时使用：

```
matrix_value("temperature")
```

在任务表表达式中，规范 QM 属性函数可以省略任务参数，隐式读取当前行任务：

```
select E = energy()
select nimag = nimag()
```

也可以显式写：

```
select E = energy(self)
```

5.4.7.2 Derived 表 `from derived(...)`

`from derived(...)` 读取另一个 `@derive` 已生成的表：

```
from derived("inventory.conformers")
from derived(inventory, table=conformers)
from derived(derive=inventory, table=conformers)
```

第一种写法使用 `derive-id.table-name` 简写；第二种和第三种把 derive id 与 table 名显式拆开。derive id 和 table 名都必须是合法标识符。

Derived 表会暴露原表的列、labels、task metadata 和 matrix metadata。数值列保留原来的 dimension 和 unit，文本列作为文本 row field 读取。因此下游 derive 可以继续使用：

```
where present(energy)
select dE = relative(energy, ref=min)
sort dE asc
```

标量区读取上游 derived value 使用 `derived()` 或 `derive_value()`：

```
let Emin = derived("inventory.min_energy")
let Emin = derived(inventory, min_energy)
let Emin = derive_value(derive=inventory, value=min_energy)
```

运行 derive pipeline 时，BaneTask 会根据 derived 表和 derived 标量引用建立 derive 依赖，并按拓扑顺序执行。指定 `--derive <name>` 时，上游 derive 依赖也会被纳入本次执行计划。发现环依赖时执行失败，例如：

```
A -> B -> A
```

上游 derived document 的 fingerprint 会进入下游 `provenance.input_fingerprint`。

5.4.7.3 原子表 `from atoms(task) as alias`

```
@derive mulliken_table
  from atoms(sp) as q
  select atom.id
  select atom.element
  select charge = q.charge.mulliken
  emit table mulliken
```

```
@end
```

`atoms()` 当前接受一个任务参数，并且必须指定简单 alias：

```
from atoms(resp_vacuum) as vac
from atoms("task-name-with-dash") as vac
```

主 atom source 会同时暴露无 alias 的公共标签：

```
atom.id
atom.element
atom.index
```

如果规范结果中有坐标标签，也会暴露：

```
atom.x
atom.y
atom.z
```

同一批标签也会暴露 alias 版本：

```
vac.atom.id
vac.atom.element
vac.atom.index
vac.atom.x
vac.atom.y
vac.atom.z
```

所有注册为 `axis=atom` 的规范属性会按以下形式暴露：

```
<alias>.<qc.atom. 后缀 >
```

例如：

```
vac.charge.mulliken
vac.charge.loewdin
vac.population.mulliken
vac.population.loewdin
vac.spin_population.mulliken
vac.spin_population.loewdin
```

属性列表中的便捷别名也会映射到相同 alias 前缀。完整名称可用以下命令查询：

```
btproc result list-properties
btproc result describe-property qc.atom.charge.mulliken
```

5.4.7.4 原子连接 `join atoms(...) ... on atom.id`

```
@derive resp2
  from atoms(resp_vacuum) as vac
  join atoms(resp_solvent) as sol on atom.id

  where present(vac.charge.mulliken)
```

```
  where present(sol.charge.mulliken)

  select atom.id
  select atom.element
  select charge.resp2 = 0.5 * vac.charge.mulliken + 0.5 * sol.charge.mulliken

  emit table resp2_charges
@end
```

当前 `join` 只支持：

```
join atoms(task) as alias on atom.id
```

连接是严格一一对应连接：

- 主表和连接表必须具有完全相同的 atom ID 集合；
- 不允许缺失 ID；
- 不允许额外 ID；
- 单个任务中不允许重复 atom ID；
- 相同 ID 的元素标签不一致时失败；
- 连接按 ID 对齐。

因此，两个任务输出的原子顺序可以不同，只要稳定 `atom.id` 一致，配对结果仍然正确。

5.4.7.5 where

```
where present(energy())
where task.program == "gaussian"
where matrix_value("temperature") >= 298.15
where abs(q.charge.mulliken) > 0.01
where present(energy()) and nimag() == 0
where task.program in [gaussian, orca]
where matrix.method matches "b3lyp*"
where not present(warning)
```

可以写多个 `where`，按声明顺序依次过滤。

条件表达式支持：

```
== != < <= > >=
and or not
in
matches
```

字符串比较支持 `==`、`!=`、`in` 和 `matches`。`matches` 使用 BaneTask 通配模式，`*` 表示任意长度文本。`in` 的右侧为列表，元素可写成 bare identifier 或字符串：

```
where task.program in [gaussian, "orca"]
```

数值比较支持全部比较运算符，并保留维度和单位检查。`and`、`or` 和 `not` 支持括号分组：

```
where present(energy()) and (task.program == gaussian or task.program == orca)
```

如果 row identifier 对应的可选表字段缺失，普通 `where` 会把该行过滤掉。推荐使用 `present(...)` 明确表达可选字段检查。

5.4.7.6 `select`

三种单项形式等价：

```
select E = energy()
select energy() as E
select task.name
```

连续声明也可写成有序列表，并在解析时展开为普通 `select`：

```
select (
  task = task.name,
  method = task.program,
  E = convert(energy(), "kcal/mol"),
)
```

列表保持声明顺序，并允许尾随逗号。

其中 bare identifier 会同时作为表达式和列名。

字符串列示例：

```
select task.name
select program = task.program
select element = atom.element
select source = "RESP2"
```

数值列示例：

```
select E = convert(energy(), "kcal/mol")
select index = row_index()
select charge = 0.5 * vac.charge.mulliken + 0.5 * sol.charge.mulliken
```

同一列的所有非空行必须保持同一类型：

- 不能混合字符串和数值；
- 数值列必须保持相同物理维度；
- 后续行会自动换算到第一条非空数值确定的列单位。

`select` 产生的列会暴露为后续表达式可读取的 row field。数值列可继续参与数值表达式，文本列可继续用于文本选择、文本比较、`group by` 和 `sort`。例如：

```
select vacuum = vac.charge.mulliken
select solvent = sol.charge.mulliken
select charge.resp2 = 0.5 * vacuum + 0.5 * solvent
```

可选数值字段缺失时，数值 cell 写成 JSON `null`。可选文本字段选择前应先用 `where present(field)` 过滤。

5.4.7.7 跨行 `select` 函数

下列函数在 `select` 顶层使用时，会对当前表的整列求值：

```
select dE = relative(E, ref=min)
select rank_id = rank(E)
select dense_id = dense_rank(E, asc=false)
select weight = boltzmann(dE, temperature=298.15)
select population = normalize(weight)
```

`relative(column, ref=...)` 从每个非缺失值中减去参考值，输出保留输入列的 dimension 和 unit。`ref` 支持：

```
min
max
first
last
```

`rank(column, asc=true)` 输出 competition rank。相同数值共享相同排名，后续排名按被占用位置递增，例如 `1, 1, 3`。`dense_rank(column, asc=true)` 输出 dense rank，相同数值共享相同排名，后续排名连续递增，例如 `1, 1, 2`。

`normalize(column)` 将每个非缺失值除以非缺失值总和，输出 dimensionless 数值；总和为零时报错。

`boltzmann(energy, temperature=298.15)` 是行级纯函数。输入为 energy dimension 时先换算为 `kcal/mol`，输出 dimensionless 权重；`temperature` 可以是无量纲数值或温度量，温度必须大于零。常见写法是先用 `relative()` 得到相对能量，再用 `boltzmann()` 和 `normalize()` 得到 population。

这些跨行函数会在 `where` 过滤后、按照 `select` 声明顺序执行，可以引用前面 `select` 产生的列。

5.4.8 分组、聚合、排序和截断

5.4.8.1 `group by`

```
group by task.program
group by matrix.method as method
group by method, basis
```

`group by` 可使用原始 row field，也可使用前面 `select` 产生的列。分组后，输出列会重置为 group key；分组前的 `select` 列不会自动进入最终表，但仍可作为 group key 或 aggregate 表达式的输入。

5.4.8.2 `aggregate` / `reduce`

`aggregate` 和 `reduce` 语义相同；格式化输出时统一写成 `aggregate`：

```
aggregate n = count()
aggregate n_with_energy = count(E)
aggregate E_min = min(E)
aggregate E_avg = mean(E)
reduce E_sum = sum(E)
```

多个聚合列也可写成列表：

```
aggregate (
  n = count(),
  E_min = min(E),
  E_mean = mean(E),
)
```

该列表按顺序展开为普通 `aggregate` 语句。

支持函数：

函数	语义
`count()`	当前 group 的行数
`count(expr)`	当前 group 中 `expr` 非 missing 的行数
`sum(expr)`	当前 group 中 `expr` 的求和
`mean(expr)` / `avg(expr)`	当前 group 中 `expr` 的平均值
`min(expr)`	当前 group 中 `expr` 的最小值
`max(expr)`	当前 group 中 `expr` 的最大值

聚合表达式必须返回标量。除 `count()` 外，聚合列保留输入表达式的 dimension 和 unit。空聚合结果写成 JSON `null`。

5.4.8.3 `sort`

```
sort E asc
sort E desc
sort task.name asc
```

未分组表的 `sort` 可使用原始 row field 和 `select` 列；分组表的 `sort` 可使用 group key 和 aggregate 列。数值排序检查 dimension 和 unit；文本排序按字符串比较。多个旧式 `sort` 可连续声明，并按声明顺序执行稳定排序；后声明的 `sort` 在最终排序优先级上更靠前。需要按通常的“左侧键优先”方式阅读时，使用多键形式：

```
sort by (
  method asc,
  E desc,
  task.name asc,
)
```

多键形式以最左侧键为最高优先级；当前键相同时再比较其右侧键。列表允许尾随逗号。

5.4.8.4 `limit`

```
limit 20
```

`limit` 接受正整数，位于所有 `sort` 之后。当前每个表流水线最多一个 `limit`。

5.4.8.5 分组汇总示例

```
@derive method_groups
  from tasks("conf_*")
  select method = matrix.method
  select E = matrix_value("energy")
  group by method
  aggregate n = count()
  aggregate E_min = min(E)
  aggregate E_mean = mean(E)
  sort E_min asc
  limit 5
  emit table method_summary
@end
```

5.4.9 表和 atom artifact sink

5.4.9.1 `emit table`

```
emit table conformer_energies
emit table conformer_energies to csv "conformer_energies.csv"
emit table conformer_energies to tsv "conformer_energies.tsv"
emit table conformer_energies to jsonl "conformer_energies.jsonl"
```

不带 `to` 时，表只写入 `.derived.json`。带 `to csv|tsv|jsonl` 时，表仍写入 `.derived.json`，并额外生成一个表文件 artifact。输出路径必须是相对路径，不能包含 `..`，实际文件写入：

```
Results/Derived/<derive-id>/<path>
```

CSV / TSV 第一行为列名；JSONL 每行对应一条表行。

输出 JSON 表包含：

- 表名；
- source 和 joins；
- 语义 axis；
- key 名；
- group key；
- sort 和 limit 元数据；
- 列名、表达式、类型、维度和单位；
- 每行的 task metadata；
- matrix 值；
- atom labels；
- 列值对象。

当前表不会直接写入 `.bane/store/records` 或 `.bane/cache/project.db`。需要进入 record / SQLite 时，仍应由结果链路或专用导入步骤完成。

5.4.9.2 `emit atomvec`

`emit atomvec` 只适用于 atom 表流水线，用于把当前逐原子数值输出为 `atom_vectors` JSON

项，同时生成默认 `.kv` artifact：

```
emit atomvec charge.resp2
emit atomvec charge.resp2 = 0.5 * vac.charge.mulliken + 0.5 *
 ↪  sol.charge.mulliken
```

默认 artifact 路径为：

```
Results/Derived/<derive-id>/<property>.atomvec.kv
```

默认 `.kv` 内容形式：

```
atoms.<atom.id>.<property>=<value>
```

`emit atomvec` 要求当前 atom pipeline 至少已经声明一个 `select`、`group by` 或 `aggregate` row field。输出表达式必须是数值，且所有非空行的 dimension 必须一致。

5.4.9.3 `emit kv`

`emit kv` 只适用于 atom 表流水线，用于指定路径生成逐原子 `.kv` artifact：

```
emit kv "resp2.values.kv" from charge.resp2
emit kv "resp2.values.kv" from 0.5 * vac.charge.mulliken + 0.5 *
 ↪  sol.charge.mulliken as charge.resp2
```

当 `from` 后是 bare column 时，property 由表达式名推断。表达式不是 bare column 时，需要写 `as <atom-property>`。

实际写入路径位于 derive 输出目录下，并自动加上 derive id 前缀：

```
Results/Derived/<derive-id>/resp2.values.kv
```

5.4.9.4 `emit artifact`

`emit artifact` 只适用于 atom 表流水线，语法为：

```
emit artifact "resp2.extxyz" from charge.resp2
emit artifact "resp2.chg" from charge.resp2
emit artifact "resp2.kv" from charge.resp2
emit artifact "resp2.extxyz" from 0.5 * vac.charge.mulliken + 0.5 *
 ↪  sol.charge.mulliken as charge.resp2
```

支持扩展名：

扩展名	artifact kind	内容
`.extxyz`	`extxyz`	element、坐标和一个逐原子数值列
`.xyz`	`extxyz`	同 `.extxyz` 渲染格式
`.chg`	`chg`	element、坐标和电荷列
`.kv`	`atom-values-kv`	`atoms.<id>.<property>=<value>`

`.extxyz`、`.xyz` 和 `.chg` 要求 atom row 中存在 `atom.element`、`atom.x`、`atom.y`、`atom.z` 标签。

artifact path 必须是相对路径，不能包含 `..`。路径中的反斜杠会规范化为 `/`。JSON 中记录 artifact 的 kind、path、property、expression、fingerprint 和 size。

5.4.9.5 多 sink 示例

```
@derive resp2
  from atoms(resp_vacuum) as vac
  join atoms(resp_solvent) as sol on atom.id

  where present(vac.charge.mulliken)
  where present(sol.charge.mulliken)

  select atom.id
  select atom.element
  select vacuum = vac.charge.mulliken
  select solvent = sol.charge.mulliken
  select charge.resp2 = 0.5 * vacuum + 0.5 * solvent

  emit atomvec charge.resp2
  emit kv "resp2.values.kv" from charge.resp2
  emit artifact "resp2.extxyz" from charge.resp2
  emit table resp2_charges
@end
```

5.4.10 表达式与函数

Derive 复用统一结果表达式，支持：

```
数字
字符串函数参数
局部变量
row field
一元 + / -
二元 + / - / * / /
函数调用
圆括号
```

5.4.10.1 规范属性函数

规范属性名和别名可以直接作为函数名：

```
energy(task)
scf_energy(task)
gibbs_corr(task)
nimag(task)
frequency(task)
oscillator_strength(task)
```

在任务表行内可以省略 task：

```
energy()
nimag()
```

规范名称、shape、unit、axis、别名和解析需求请使用：

```
btproc result list-properties
btproc result describe-property qc.energy.scf
```

当前规范属性目录暴露的主要类别包括：

- energy；
- thermochemistry；
- vibration；
- optimization step；
- excited state；
- atom charge / population / spin population。

5.4.10.2 `energy()` / `single_point()`

默认读取 primary energy：

```
energy(sp)
single_point(sp)
```

可以用 `kind=` 显式指定：

```
energy(sp, kind=scf)
energy(sp, kind=correlated)
energy(sp, kind=double_hybrid)
energy(sp, kind=composite)
energy(sp, kind=composite_zero_k)
energy(freq, kind=gibbs)
energy(sp, kind=primary)
```

支持值：

```
scf
correlated / post_scf / post_hf / posthf
double_hybrid / doublehybrid / double-hybrid / dh
composite / cbs_qb3 / cbsqb3
composite_zero_k / composite_0k / composite_zero / zero_k_composite /
 ↪  cbs_qb3_0k / cbsqb3_0k
gibbs / free
primary / electronic
```

`single_point()` 与 `energy()` 使用相同的查询规则。

5.4.10.2.1 规范能量记录选择

当一个任务的规范结果中包含多个原生能量项时，可以按能量本身的语义选择记录：

```
let E2ab = energy(ccsdt, method=MP2, quantity=correlation, spin="alpha-beta",
 ↪  reference=hf, state=0)
let Eccsd = energy(ccsdt, method=CCSD, quantity=total, reference=hf)
```

```
let Et = energy(ccsdt, method="CCSD(T)", quantity=total, reference=hf)
```

可用 selector 如下：

selector	含义
`method=`	电子结构方法，例如 `HF`、`B3LYP`、`MP2`、`MP3`、`CCSD`、`CCSD(T)`、`D3(BJ)`
`quantity=` / `energy_quantity=`	`total`、`correlation`、`perturbative_increment` 或 `correction`
`spin=` / `spin_component=`	`alpha-alpha`、`alpha-beta`、`beta-beta` 或 `none`；接受 `aa`、`ab`、`bb`、`os` 等等价写法
`reference=` / `orbital_reference=`	`hf` 或 `ks`，用于区分 Hartree-Fock 轨道和 Kohn-Sham 轨道上的相关能
`state=` / `state_index=`	非负电子态编号；基态为 `0`
`energy_kind=`	选填的粗粒度能量类别
`job=` / `job_index=`	规范结果中的 job 序号
`step=` / `step_index=`	step 序号；同时给出 `job=` 时为该 job 内局部序号，否则为文档全局序号
`final=`	`true` 取最后一个匹配项；`false` 要求 selector 只匹配一个记录

规范能量查询至少需要一个身份 selector：`method`、`quantity`、`spin`、`reference`、`state` 或 `energy_kind`。只给出 `job`、`step` 或 `final` 不足以确定能量含义。

同一个方法产生的能量使用同一套 selector，不因它出现在何种父任务中而改变名称。例如，CCSD(T) 输出中的 MP2 初始相关能仍使用 `method=MP2`。组合方法中的差值由表达式计算：

```
let E2ss = energy(ccsdt, method=MP2, quantity=correlation, spin="alpha-alpha",
↪  reference=hf) + energy(ccsdt, method=MP2, quantity=correlation,
↪  spin="beta-beta", reference=hf)
let dCCSD_MP2 = energy(ccsdt, method=CCSD, quantity=total, reference=hf) -
↪  energy(ccsdt, method=MP2, quantity=total, reference=hf)
let triples = energy(ccsdt, method="CCSD(T)", quantity=total, reference=hf) -
↪  energy(ccsdt, method=CCSD, quantity=total, reference=hf)
```

`result()` 可以直接查询同一属性：

```
let E2ab = result(ccsdt, "qc.energy.value", method=MP2, quantity=correlation,
↪  spin="alpha-beta", reference=hf)
```

只使用 `kind=` 而不提供规范身份 selector 时，`energy()` 读取前述摘要能量类别。规范查询中同时提供 `method=` 等身份 selector 时，`kind=` 等价于粗粒度的 `energy_kind=` 过滤条件。

5.4.10.3 `derived()` / `derive_value()`

读取上游 derived document 中的标量 value：

```
let Emin = derived("inventory.min_energy")
let Emin = derived(inventory, min_energy)
let Emin = derive_value(derive=inventory, value=min_energy)
```

`derived("derive.value")` 是简写形式。双参数写法中，第一个参数是 derive id，第二个参数是 value 名。命名参数写法必须同时给出 `derive=` 和 `value=`；`id=` 可替代 `derive=`，`name=` 或 `scalar=` 可替代 `value=`。

被读取的上游 derived document 必须包含对应 scalar value。该引用会进入 derive 依赖分析和 fingerprint。

5.4.10.4 `result(task, property)`

使用规范属性名或别名查询统一结果层：

```
result(opt, "qc.optimization.energy")
result(freq, "qc.vibration.frequency")
result(td, "oscillator_strength")
```

`property` 必须是统一量化结果查询中可识别的属性名或别名。该函数不会隐式回退到 `.bane.result.kv`。

5.4.10.5 通用数值与热力学函数

5.4.10.5.1 `exp()` / `log()` / `sqrt()` / `pow()`

这些函数只接受无单位标量：

```
let decay = exp(-1.63)
let restored = log(exp(2))
let root = sqrt(4)
let power = pow(2, 3)
let named_power = pow(base=2, exponent=3)
```

`log()` 的参数必须大于零，`sqrt()` 的参数不得小于零。运算结果必须是有限实数。

5.4.10.5.2 `cbs(low, high, alpha, beta)`

`cbs()` 执行通用二点线性外推：

```
(alpha * high - beta * low) / (alpha - beta)
```

`low` 与 `high` 必须具有相同维度；单位不同时会先换算到 `low` 的单位。`alpha` 与 `beta` 是无单位且互不相等的标量。例如，指数型 HF 外推可以写成：

```
let E_hf_cbs = cbs(E_hf_tz, E_hf_qz, 1, exp(-1.63))
```

外推函数不识别方法名或基组名，输入值及参数由任务流程和 derive 表达式确定。

5.4.10.5.3 `thermo(task, quantity=..., ...)`

`thermo()` 使用任务的规范化量化结果，从最终振动分析、对应几何、原子质量、分子多重度和对称性信息重新计算热力学量。该函数不复用程序输出中已经打印的热校正值，适合在组合方法电子能确定后，用统一的热力学模型和频率校正因子重新组装焓与 Gibbs 自由能。

`thermochemistry()` 和 `rethermo()` 是等价别名。task 可以作为第一个位置参数，也可以写成 `task=`；两种写法不能同时使用，且位置参数最多一个。`quantity=` 必须显式给出：

```
let E_composite = E_hf_cbs + E_correlation
let G_298 = thermo(freq, quantity=gibbs_free_energy, model=rrho,
 ↪  temperature=298.15, pressure=1.0, zpe_scale=0.9766, enthalpy_scale=0.9791,
 ↪  entropy_scale=0.9647, electronic_energy=E_composite)
let G_298_1M = thermo(freq, quantity=gibbs_free_energy, model=rrho,
 ↪  temperature=298.15, pressure=1.0, concentration=1.0, zpe_scale=0.9766,
 ↪  enthalpy_scale=0.9791, entropy_scale=0.9647,
 ↪  electronic_energy=E_composite)
emit free_energy_298_kcal_mol = convert(G_298, "kcal/mol")
emit free_energy_298_1M_kcal_mol = convert(G_298_1M, "kcal/mol")
```

`thermo()` 调用可按括号平衡规则跨多行，并允许尾随逗号。相同结果源和相同参数在一个 derive 中只计算一次，后续 quantity 查询复用缓存结果；重复参数包可用 `profile`，多个 quantity 可用 `bind thermo(...) to (...)`。

热力学模型

模型、转子类型、质量策略和 `quantity=` 选择器在匹配前会转为小写，并移除连字符、下划线和空白。因此 `grimme_qrrho`、`Grimme-qRRHO` 与 `grimme qrrho` 等价。表中首先给出推荐写法，再列出常用短名。

`model=`	含义
`rrho` / `harmonic`	理想气体刚性转子-谐振子模型。论文或补充材料给出常规谐振热力学缩放因子时应选此项。
`truhlar_qrrho` / `truhlar`	Truhlar 准 RRHO；低于 `low_frequency_cutoff` 的正频率按截止频率处理。
`grimme_qrrho` / `grimme` / `qrrho`	Grimme 准 RRHO；低频振动熵在谐振子与自由转子之间插值。省略 `model=` 时使用此模型。
`minenkov_qrrho` / `minenkov`	Minenkov 准 RRHO；振动内能和熵均进行低频插值。该模型没有可唯一分离的 ZPVE，因此不能查询 `zero_point_energy`。

参数

参数	默认值	说明
`temperature=`	`298.15`	温度，接受 K 维度标量或按 K 解释的无单位标量，必须大于零。
`pressure=`	`1.0`	压力，接受 atm 维度标量或按 atm 解释的无单位标量，必须大于零。

参数	默认值	说明
`concentration=` / `standard_state_concentration=`	未设置	目标浓度标准态，使用正数并按 mol/L 解释。未设置时保持由 `pressure=` 定义的理想气体压力标准态；设置后只修正平动熵、配分函数和 Gibbs 自由能，U、H、Cv、Cp 不变。该参数要求 `include_translation=true`。完整别名为 `standard_state_concentration_mol_per_liter=`。
`zpe_scale=` / `zero_point_scale=`	`1.0`	仅用于 ZPVE 的振动频率缩放因子。
`enthalpy_scale=` / `thermal_enthalpy_scale=`	`1.0`	用于振动热内能和热焓项的频率缩放因子，不缩放平动、转动或 `RT` 项。
`entropy_scale=`	`1.0`	用于振动熵项的频率缩放因子，不缩放平动、转动或电子熵。
`heat_capacity_scale=`	`1.0`	用于振动热容项的频率缩放因子。
`low_frequency_cutoff=` / `low_frequency_cutoff_cm1=`	`100.0`	Truhlar qRRHO 截止频率，单位 cm^-1。
`interpolation_frequency=` / `interpolation_frequency_cm1=`	`100.0`	Grimme 和 Minenkov qRRHO 插值中点，单位 cm^-1。
`imaginary_frequency_threshold=` / `imaginary_frequency_threshold_cm1=`	`0.0`	绝对值不超过该阈值的负频率按正频率处理；其余非正频率不进入振动求和。单位 cm^-1。
`symmetry=` / `rotational_symmetry_number=`	自动	显式指定正整数转动对称数。
`infer_symmetry=` / `infer_rotational_symmetry_number=`	`true`	未显式给出对称数时，先使用结果中的点群信息，再从参考几何识别点群；两者都不能可靠确定时使用 1 并记录诊断。
`rotor=` / `rotor_kind=`	`auto`	`auto`、`monatomic` / `atom`、`linear` 或 `nonlinear`。`auto` 根据几何主惯性矩判断。
`include_translation=`	`true`	是否计入理想气体平动贡献。
`include_rotation=`	`true`	是否计入刚性转子贡献。
`free_rotor_average_moment=` / `free_rotor_average_moment_kg_m2=`	`1.0e-44`	Grimme / Minenkov 自由转子插值使用的平均惯性矩，单位 kg m^2。

参数	默认值	说明
`mass_policy=`	`most_abundant_isotope`	原子质量策略。`most_abundant_isotope` 还接受 `mostabundant`、`isotope`；`standard_atomic_weight` 还接受 `standardweight`、`average`。
`electronic_energy=`	任务电子能	覆盖热力学总量使用的电子能，必须是 Energy 维度。组合方法应在这里传入已经完成外推、相关修正、色散修正和相对论修正的电子能。

四个频率缩放因子必须为正数。`electronic_energy=` 只改变总内能、总焓和总 Gibbs 自由能，不改变 ZPVE、热校正、熵或热容。

指定浓度标准态时，程序按当前温度和压力求理想气体浓度 `c_gas`，再施加 `Delta G = R*T*ln(c_target/c_gas)` 和相应的平动熵修正。`pressure=` 仍用于定义换算起点，因此同一个 `concentration=` 在不同温度或压力下得到的修正可能不同。

可查询量

`quantity=`	返回内容	维度和默认单位
`electronic_energy` / `electronic`	实际采用的电子能	Energy, Hartree
`zero_point_energy` / `zero_point` / `zpe`	缩放后的 ZPVE	Energy, Hartree
`thermal_energy` / `internal_energy`	电子能加热内能修正	Energy, Hartree
`thermal_correction` / `internal_energy_correction`	含 ZPVE 的热内能修正	Energy, Hartree
`enthalpy`	电子能加总焓修正	Energy, Hartree
`enthalpy_correction`	ZPVE、平动、转动、振动热项、电子态热项和 `pV` 项之和	Energy, Hartree
`entropy`	总熵	Entropy, J/(mol*K)
`entropy_correction` / `temperature_times_entropy` / `ts`	`T*S`	Energy, Hartree
`gibbs_free_energy` / `gibbs_energy` / `free_energy` / `gibbs`	`electronic_energy + gibbs_correction`	Energy, Hartree
`gibbs_correction` / `free_energy_correction`	`enthalpy_correction - T*S`	Energy, Hartree

quantity=	返回内容	维度和默认单位
cv / constant_volume_heat_capacity	定容热容	HeatCapacity, J/(mol*K)
cp / constant_pressure_heat_capacity	定压热容	HeatCapacity, J/(mol*K)
temperature	实际温度	Temperature, K
pressure	实际压力	Pressure, atm
ideal_gas_concentration / pressure_concentration	当前温度和压力对应的理想气体浓度	Scalar，mol/L
standard_state_concentration / concentration	显式设置的目标浓度；未设置时不可查询	Scalar，mol/L
standard_state_gibbs_correction / concentration_correction	从压力标准态换算到目标浓度标准态的 Gibbs 修正	Energy, Hartree
total_mass / mass	总质量	Mass，amu
principal_moment_a / moment_a / ia、principal_moment_b / moment_b / ib、principal_moment_c / moment_c / ic	按升序排列的三个主惯性矩	Scalar, amu*angstrom^2
rotational_symmetry_number / symmetry_number / symmetry	实际转动对称数	Dimensionless
point_group	实际采用或由几何识别的点群；无可用点群时不可查询	Text scalar
symmetry_source / rotational_symmetry_source	对称数来源：explicit、metadata、geometry 或 default	Text scalar
input_frequency_count	输入振动频率数	Dimensionless
used_frequency_count	进入求和的正频率数	Dimensionless
ignored_nonpositive_frequency_count	被忽略的非正频率数	Dimensionless
corrected_imaginary_frequency_count	按阈值反射为正值的负频率数	Dimensionless
translation_internal_energy、rotation_internal_energy、vibration_internal_energy、electronic_internal_energy	各分量的内能贡献；也可用后缀 _u	Energy, Hartree

quantity=	返回内容	维度和默认单位
translation_enthalpy、rotation_enthalpy、vibration_enthalpy、electronic_enthalpy	各分量的焓贡献；也可用后缀 _h	Energy, Hartree
translation_entropy、rotation_entropy、vibration_entropy、electronic_entropy	各分量的熵贡献；也可用后缀 _s	Entropy, J/(mol*K)
translation_cv、rotation_cv、vibration_cv、electronic_cv	各分量的定容热容	HeatCapacity, J/(mol*K)
translation_cp、rotation_cp、vibration_cp、electronic_cp	各分量的定压热容	HeatCapacity, J/(mol*K)
translation_logq、rotation_logq、vibration_logq、electronic_logq	各分量的配分函数自然对数；完整后缀为 _log_partition_function	Dimensionless
log_vibrational_partition_function_v0 / vibrational_logq_v0	以振动基态为零点的振动配分函数自然对数	Dimensionless
log_vibrational_partition_function_bottom / vibrational_logq_bottom	以势阱底为零点的振动配分函数自然对数	Dimensionless
log_partition_function_v0 / logq_v0	以振动基态为零点的总配分函数自然对数	Dimensionless
log_partition_function_bottom / logq_bottom	以势阱底为零点的总配分函数自然对数	Dimensionless
model、rotor_kind / rotor、implementation	实际模型、转子类型和实现名称	Text scalar

热焓拆分通常写成：

```
let ZPVE = thermo(freq, quantity=zero_point_energy, model=rrho,
 ↪  zpe_scale=0.9766, enthalpy_scale=0.9791, entropy_scale=0.9647)
let Hcorr = thermo(freq, quantity=enthalpy_correction, model=rrho,
 ↪  zpe_scale=0.9766, enthalpy_scale=0.9791, entropy_scale=0.9647)
let Hthermal_without_ZPVE = Hcorr - ZPVE
let TS = thermo(freq, quantity=entropy_correction, model=rrho,
 ↪  zpe_scale=0.9766, enthalpy_scale=0.9791, entropy_scale=0.9647)
let Gcorr = Hcorr - TS
```

计算要求任务结果中存在最终振动分析、与其关联的几何和有效化学体系。默认电子简并度取分子多重度；默认原子质量取最丰同位素；默认依次根据结果点群和参考几何推断转动对称数。用于稳定点自由能时，宜先用 `require nimag(freq) == 0` 排除虚频结构。

`harmonic_zpe()` 和 `rrho_enthalpy()` 是只依赖传入频率序列的简化数值函数，不读取

几何、质量、转动对称数、压力或电子多重度，也不计算总熵与自由能。需要完整 RRHO / qRRHO 热力学或组合电子能自由能时，应使用 `thermo()`。

5.4.10.5.4 `harmonic_zpe(frequencies, scale=1)`

`harmonic_zpe()` 从 cm^-1 频率序列计算谐振零点能，只累加正频率，并以 Hartree 返回：

```
let frequencies = result(freq, "qc.vibration.frequency")
let zpe_scaled = harmonic_zpe(frequencies, scale=0.9766)
```

`zpe(task)` 读取程序输出中已经报告的零点能；`harmonic_zpe(series)` 则根据频率重新计算，两者输入含义不同。

5.4.10.5.5 `rrho_enthalpy(frequencies, natoms, linear=false, temperature=298.15, scale=1)`

`rrho_enthalpy()` 按理想气体刚性转子-谐振子模型计算平动、转动和振动热焓修正，不包含电子能和零点能，返回单位为 Hartree：

```
let h_thermal = rrho_enthalpy(frequencies, natoms=3, linear=false,
↪  temperature=298.15, scale=0.9791)
```

只使用正频率。单原子的平动与转动项为 `2.5 RT`，线性分子为 `3.5 RT`，非线性分子为 `4.0 RT`。`temperature` 可以是以 K 为单位的温度标量，也可以是按 K 解释的无单位标量；`scale` 必须为正数。

5.4.10.6 `kv(task, key)`

读取轻量结果快照中的数值字段：

```
kv(opt, "custom.score")
```

`kv()` 仅适用于数值。其维度通过旧 key 名做有限推断；无法识别时视为 dimensionless。新写法优先使用规范属性和 `result()`。

5.4.10.7 `convert(value, unit)`

```
convert(energy(sp), "kcal/mol")
convert(result(opt, "qc.optimization.energy"), "kJ/mol")
```

标量和序列都可以换算。换算必须满足结果层支持的同维度单位规则。

5.4.10.8 `quantity(value, dimension=..., unit=...)`

`quantity()` 把有限、无单位的数值标量显式构造成物理量：

```
let proton_free_energy = quantity(-270.29, dimension=energy, unit="kcal/mol")
let temperature = quantity(298.15, dimension=temperature, unit="K")
let cutoff = quantity(100, dimension=frequency, unit="cm-1")
```

也可全部使用位置参数：

```
let RTln10 = quantity([RTln10_kcal], energy, "kcal/mol")
```

支持的 dimension 包括 `dimensionless`、`energy`、`force`、`length`、`frequency`、`spectral_coordinate`、`angle`、`step`、`temperature`、`pressure`、`mass`、`entropy`、`heat_capacity`

和 dipole_moment。unit 必须与 dimension 匹配；非无量纲输入、带现有单位的输入或非有限数值直接报错。quantity() 用于构造，convert() 与 emit ... unit ... 用于已有物理量的单位换算，二者不能互相替代。

5.4.10.9 min() / max()

```
max(result(opt, "qc.optimization.max_force"))
min(result(scan, "qc.optimization.energy"))
```

标量输入原样返回；序列输入忽略 missing sample，并归约为标量。空序列或全部 missing 时失败。

表级求和和均值使用 aggregate sum(...)、aggregate mean(...) 或 aggregate avg(...)。

5.4.10.10 sort()

sort() 对数值序列进行稳定排序，并返回保持 key、value、threshold 对齐的新序列：

```
let sorted_energy = sort(result(td, "qc.excited_state.energy"))
let descending_energy = sort(result(td, "qc.excited_state.energy"), asc=false)
```

默认按序列自身数值升序排序；asc=false 改为降序。missing sample 始终排在末尾，相同排序值保持原始顺序。

也可以用同一 axis、同一组 entity key 的另一条数值序列作为排序键：

```
let energies_by_multiplicity = sort(
  result(td, "qc.excited_state.energy"),
  by=result(td, "qc.excited_state.multiplicity")
)
```

by= 序列必须与被排序序列具有相同 axis，且 entity key 和顺序完全一致，否则求值失败。函数调用可按本节开头所述的逻辑语句规则跨行书写。

5.4.10.11 filter()

filter() 按数值条件筛选序列，并保持 key、value、threshold 的对应关系：

```
let singlet_energy = filter(result(td, "qc.excited_state.excitation_energy"),
↪  by=result(td, "qc.excited_state.multiplicity"), eq=1)
let triplet_energy = filter(result(td, "qc.excited_state.excitation_energy"),
↪  by=result(td, "qc.excited_state.multiplicity"), eq=3)
```

第一个位置参数是返回值序列。by= 指定用于判定的选择器序列；省略 by= 时，以返回值序列自身作为选择器。选择器序列与返回值序列必须具有相同 axis，且 entity key 和顺序完全一致。

每次调用必须且只能提供一个比较参数：

参数	条件
eq=	等于
ne=	不等于

参数	条件
`lt=`	小于
`le=`	小于或等于
`gt=`	大于
`ge=`	大于或等于

选择器中的 missing sample 不进入输出。比较值必须与选择器序列维度一致；不带单位的数字按选择器当前单位解释。筛选结果保持原顺序，可继续传给 `sort()`、`min()`、`max()`、`at()` 或 `nth()`。

5.4.10.12 `at()` / `nth()`

`at()` 和 `nth()` 从数值序列中提取一个标量：

```
let first_zero_based = at(series, 0)
let first_one_based = nth(series, 1)
```

- `at(series, index)` 使用零基位置；
- `nth(series, n)` 使用一基位置；
- 位置必须是非负整数，`nth()` 的位置必须至少为 1；
- 越界或选中 missing sample 时求值失败；
- 返回标量保留原序列的维度和单位。

激发态能级可以按多重度筛选、按能量排序，再提取 S1、T1 和 T2：

```
@derive excited_levels
  let E = result(soc, "qc.excited_state.excitation_energy")
  let M = result(soc, "qc.excited_state.multiplicity")
  let S = sort(filter(E, by=M, eq=1))
  let T = sort(filter(E, by=M, eq=3))
  emit S1 = nth(S, 1)
  emit T1 = nth(T, 1)
  emit T2 = nth(T, 2)
  emit delta_S1_T1 = abs(nth(S, 1) - nth(T, 1))
  emit delta_S1_T2 = abs(nth(S, 1) - nth(T, 2))
@end
```

`let`、`emit` 和函数调用均可按本章开头的逻辑续行规则跨多行。

5.4.10.13 `abs()`

```
abs(delta_e)
abs(q.charge.mulliken)
```

支持标量和序列，保留原维度与单位。

5.4.10.14 `present()` / `exists()`

两者为别名，主要用于表行中的可选字段：

```
where present(energy())
where exists(vac.charge.mulliken)
```

对 atom table 中已声明但当前 atom 缺失的属性返回 false。对任务表中的属性查询失败，`present()` 会在 row context 中返回 false，便于过滤没有该结果的任务。

在 derive 顶层标量区，缺少 row context 时，`present()` 不会屏蔽任意结果解析错误。

5.4.10.15 `coalesce()`

```
select charge = coalesce(q.charge.loewdin, q.charge.mulliken)
```

按顺序返回第一个可用表达式。当前主要处理 table row 中的 missing numeric field，不处理未知属性错误。

5.4.10.16 `row_index()` / `task_index()`

```
select index = row_index()
```

两者当前等价，只能用于表行表达式。返回源表中的零基序号。经过 `where` 后仍保留原 source index。

5.4.10.17 `matrix_value(name)`

```
select temperature = matrix_value("temperature")
```

只在任务表中可用。要求对应 matrix 值存在且是纯数值。

5.4.11 算术和单位规则

5.4.11.1 加减

标量加减要求物理维度兼容：

```
energy(a) - energy(b)
```

相同维度不同单位会自动换算。无单位数字与带量纲量相加减时，结果采用带量纲一侧的维度和单位：

```
energy(a) + 0
```

这种写法适合 `0` 等中性常数。非零物理常数应使用 `quantity()` 明确声明 dimension 和 unit。

序列加减要求两侧 key 完全一致、长度一致、维度一致。任一侧 sample missing 时，结果 sample 为 missing。

5.4.11.2 乘法

带量纲值可以与无量纲标量相乘：

```
0.5 * energy(a)
2 * q.charge.mulliken
```

两个带量纲值相乘会报错。

5.4.11.3 除法

带量纲值可除以无量纲标量：

```
energy(a) / 2
```

同维度标量相除会得到 dimensionless：

```
energy(a) / energy(b)
```

一般复合量纲除法和两个序列之间的逐点乘除当前不支持。

5.4.11.4 原子向量计算

atom source 已把每个 atom property 投影为当前行的标量，因此：

```
0.5 * vac.charge.mulliken + 0.5 * sol.charge.mulliken
```

是逐行标量运算。原子配对由前面的 `join ... on atom.id` 明确完成。

5.4.12 输出格式和 provenance

最小结构示意：

```
{
  "schema": "banetask.derived.v1",
  "derive_id": "activation",
  "derive_name": "activation",
  "plan": {
    "dependencies": ["sp_R", "freq_R", "sp_TS", "freq_TS"],
    "derive_dependencies": [],
    "requirements": {}
  },
  "provenance": {
    "dsl_fingerprint": "...",
    "input_fingerprint": "...",
    "document_fingerprint": "...",
    "inputs": []
  },
  "values": [
    {
      "name": "delta_g",
      "expression": "convert(G_ts - G_r, \"kcal/mol\")",
      "result": {
        "shape": "scalar",
        "value": 12.3,
        "dimension": "energy",
        "unit": "kcal/mol"
      }
    }
  ],
  "report_lines": [
    {
```

```
      "label": "Delta G activation",
      "expression": "delta_g",
      "result": {
        "shape": "scalar",
        "value": 12.3,
        "dimension": "energy",
        "unit": "kcal/mol"
      }
    }
  ],
  "report_sinks": [
    {
      "scope": "case-root",
      "path": "Report_activation.txt"
    }
  ],
  "atom_vectors": [],
  "artifacts": [
    {
      "kind": "derive-report",
      "scope": "case-root",
      "path": "Report_activation.txt",
      "fingerprint": "..."
    }
  ],
  "tables": []
}
```

report_lines 保存每条 **report** 的 label、表达式和标量结果；**report_sinks** 保存 **emit report** 声明。**artifacts** 中的普通 derived artifact 使用 **scope: "derived"** 并写入 **Results/Derived/<derive-id>/...**；根目录报告使用 **scope: "case-root"** 并写入 case / project 根目录。

表输出会记录：

```
source
joins
axis
key
columns
group_keys
sorts
limit
rows[].task/program/template_task/struc_id
rows[].matrix
rows[].labels
rows[].values
```

写文件使用临时文件加替换的原子安装方式，避免写入中途留下半截 JSON 或半截 artifact。

provenance.input_fingerprint 会纳入 derive DSL fingerprint、依赖任务、derive 依

赖、上游 derived document fingerprint、结果解析需求、任务元数据、matrix 值和可发现的结果输入文件 fingerprint。已存在的 derive 输出在 fingerprint 一致时会跳过；artifact 缺失或 artifact fingerprint 不一致时会重新写入。

5.4.13 完整示例

5.4.13.1 活化自由能

```
@derive activation
  let G_r = single_point(sp_R) + gibbs_corr(freq_R)
  let G_ts = single_point(sp_TS) + gibbs_corr(freq_TS)

  require nimag(freq_R) == 0
  require nimag(freq_TS) == 1

  emit G_r = convert(G_r, "kcal/mol")
  emit G_ts = convert(G_ts, "kcal/mol")
  emit delta_g = convert(G_ts - G_r, "kcal/mol")
@end
```

5.4.13.2 根目录快速报告

```
@derive pka_summary
  let G_acid_aq = free_energy(acid_freq) + single_point(acid_sp)
  let G_anion_aq = free_energy(anion_freq) + single_point(anion_sp)
  let G_Hplus_aq = -270.29
  let DG_deprot = convert(G_anion_aq + G_Hplus_aq - G_acid_aq, "kcal/mol")
  emit pka = DG_deprot / 1.364

  report "Final aqueous free energy of acid AH" = G_acid_aq
  report "Final aqueous free energy of anion A-" = G_anion_aq
  report "Aqueous proton free energy" = G_Hplus_aq
  report "Delta G deprotonation" = DG_deprot
  report "pKa" = pka
  emit report to root "Report_pka_summary.txt"
@end
```

该写法同时保留规范 JSON：

```
Results/Derived/pka_summary.derived.json
```

并在根目录生成便于人工归档和快速查看的：

```
Report_pka_summary.txt
```

5.4.13.3 批量任务结果表、相对能量和 population

```
@derive conformer_inventory
  from tasks("conf_*", program="gaussian")

  where present(energy()) and nimag() == 0
```

```
  select task.name
  select program = task.program
  select G = convert(energy(kind=gibbs), "kcal/mol")
  select dG = relative(G, ref=min)
  select rank = rank(G)
  select weight = boltzmann(dG, temperature=298.15)
  select population = normalize(weight)

  sort dG asc
  limit 20
  emit table conformers
  emit table conformers to csv "conformers.csv"
@end
```

5.4.13.4 读取上游 derived table

```
@derive ranked_conformers
  from derived("conformer_inventory.conformers")
  where present(G)

  select task.name
  select dG = relative(G, ref=min)
  select population_from_inventory = population

  sort dG asc
  emit table ranked to tsv "ranked.tsv"
@end
```

5.4.13.5 读取上游 derived value

```
@derive activation_report
  let dg = derived("activation.delta_g")
  emit dg_kcal = convert(dg, "kcal/mol")
  emit value dg_kcal to kv "activation.values.kv"
@end
```

5.4.13.6 按方法分组汇总

```
@derive method_summary
  from tasks("conf_*")
  where present(energy())

  select method = matrix.method
  select E = convert(energy(), "kcal/mol")

  group by method
  aggregate n = count()
  aggregate E_min = min(E)
  aggregate E_mean = mean(E)
```

```
  sort E_min asc
  emit table method_summary
@end
```

5.4.13.7 RESP2 风格电荷平均和 artifact 输出

```
@derive resp2
  from atoms(resp_vacuum) as vac
  join atoms(resp_solvent) as sol on atom.id

  where present(vac.charge.mulliken)
  where present(sol.charge.mulliken)

  select atom.id
  select atom.element
  select vacuum = vac.charge.mulliken
  select solvent = sol.charge.mulliken
  select charge.resp2 = 0.5 * vacuum + 0.5 * solvent

  emit atomvec charge.resp2
  emit kv "resp2.values.kv" from charge.resp2
  emit artifact "resp2.extxyz" from charge.resp2
  emit artifact "resp2.chg" from charge.resp2
  emit table resp2_charges
@end
```

5.5 独立 Derive 脚本

`.btder` 用于把一个 Derive 声明从主 `.bt` 文件中拆出，形成可单独复制、双击或从命令行执行的结果计算脚本。它读取已有 case 结果，不替代描述任务和 workflow 的 `.bt` 文件。

5.5.1 文件配对

显式打开 `.btder` 时，BaneTask 使用脚本所在目录作为配对目录，并只检查当前层：

```
case/
  case.bt
  activation.btder
```

配对目录必须恰好包含一个 `.bt` 文件。不递归检查子目录；`.projbt`、`.btfrag` 和 `.btlib` 不参与计数。没有 `.bt` 或存在多个 `.bt` 时直接报错。配对后的 `.bt` 提供任务定义、顶层 `define:`、matrix 展开结果、已有 `@derive` 依赖和 case 根目录；输出仍写入该 case 的 `Results/`。

每个独立脚本只允许声明一个 derive。变量替换使用配对 `.bt` 已解析的 `define:` 值。

5.5.2 文件格式

推荐直接写 `@derive` 块内部语句，不写外层头和 `@end`：

```
# activation.btder
let G_r = single_point(sp_R) + gibbs_corr(freq_R)
let G_ts = single_point(sp_TS) + gibbs_corr(freq_TS)

require nimag(freq_R) == 0
require nimag(freq_TS) == 1

emit delta_g = convert(G_ts - G_r, "kcal/mol")
```

省略头部时，文件名主体同时作为 derive name 和 id，因此必须满足 derive 标识符规则。也可写完整声明：

```
@derive activation
  let G_r = single_point(sp_R) + gibbs_corr(freq_R)
  let G_ts = single_point(sp_TS) + gibbs_corr(freq_TS)
  emit delta_g = convert(G_ts - G_r, "kcal/mol")
@end
```

独立文件开头也接受 `derive activation`；末尾 `@end` 可省略。显式名称优先于文件名。

5.5.3 执行与限制

```
banetask activation.btder
banetask activation.btder --explain-derives
banetask activation.btder --rerender-derives
banetask activation.btder --derive activation
```

该入口进入严格的 derives-only 路径：不准备任务，只执行当前 derive 及其必要依赖。配对 `.bt` 中同 id 的项目级 derive 会在本次执行视图中被独立脚本覆盖，但磁盘文件不修改。`--derive` 只能选择当前脚本的名称或 id；`.btder` 不能与 plot-only、plot explain、plot selector 或 plot rerender 选项混用。

Windows 安装器会把 `.btder` 的打开命令关联到 `banetask.exe "%1"`。分发脚本时应同时说明所需任务名、derived id、结果属性和运行环境，并把脚本放入恰好包含一个 `.bt` 的 case 目录。

5.6 Plot DSL

5.6.1 用途

BaneTask Plot DSL 用于在 `.bt` / `.projbt` 中声明由任务结果生成的可归档图形 artifact。用户在任务文件中写清数据来源、单位和输出要求，BaneTask 会读取任务结果，求值表达式，生成数据表、渲染源文件和最终输出。PNG、SVG、PDF 与文本终端由 gnuplot 渲染；`barrier`、`frontier` / `excited` 还可由 BaneTask 原生生成 ChemDraw CDXML 文档。若未显式指定 `output size`，渲染器会使用各图种的默认画布策略；其中 `barrier` 会按反应路径点数调整宽度，`frontier` / `excited` 会按比较列数和每列能级数量自动调整画布尺寸。Plot DSL 全部图种使用统一的较大默认字号，用于标题、坐标轴标签、刻度、图例和图内标注。除 `heatmap` 外，默认渲染会加粗坐标外框、数据曲线、误差棒、柱图边线和能级图箭头；`heatmap` 使用矩阵色块的轻量外框策略，避免色图外框过重。

典型输出包括一份公开图文件和一份可复现 artifact 目录：

```
Results/Plots/<plot-file>
Results/Artifacts/plots/<safe-plot-id>/
  data.tsv
  render.gp 或 render.cdxml
  spec.plotdsl
  manifest.json
```

仓库中的 **examples/plotdsl_toy_formaldehyde.bt** 与同名 **.xyz** 提供了一个用于人工试跑的甲醛 toy workflow，覆盖 convergence、barrier、curve、spectrum、scatter、heatmap、histogram、bar、errorbar、frontier 与 excited 图。

图类型包括：

- **barrier**：跨任务标量构成的能垒或反应路径图；
- **convergence**：单任务优化步、能量和收敛指标曲线；
- **curve**：普通序列或 matrix / variant 聚合曲线；
- **spectrum**：IR、Raman、UV-Vis 等 stick / Gaussian broadened spectrum；
- **scatter** / **correlation**：相关图、benchmark 散点图和线性拟合；
- **heatmap** / **matrixmap**：二维 scan、method × basis、fragment × fragment 等 z-map；
- **histogram** / **hist**：数值分布直方图；
- **bar** / **barchart**：类别柱图；
- **errorbar** / **errorbars**：带 y error bar 的统计图；
- **frontier** / **fmo** / **orbital-level** / **orbital_levels** / **orbitallevels**：前线轨道能级图，用横线表示轨道能级，并在占据轨道上绘制电子箭头；
- **excited** / **excited-state** / **excited_state** / **excited-level** / **excited_levels** / **excitedlevels** / **eslevel** / **eslevels**：激发态能级图，用横线表示各 root 的激发态能量，并可绘制跨任务 root 箭头。

5.6.2 两类声明

5.6.2.1 任务内 %plot

任务内绘图默认以当前任务为结果上下文，图 ID 为 **<task>:<plot-name>**：

```
$opt_TS
  %source origin
  %program gaussian
  %keywords "opt b3lyp/def2svp"

  %plot convergence conv
    from self
    x step.ordinal label "Step"
    y step.energy label "Energy" unit hartree
    y step.convergence(max_force) label "Max force"
    y step.convergence(rms_force) label "RMS force"
    y step.convergence(max_displacement) label "Max displacement"
    y step.convergence(rms_displacement) label "RMS displacement"
    output "Results/Plots/[taskname]_conv.png"
```

`%plot` 没有单独的结束标记。它与其他任务子块相同，由下一个同级 `%...` 指令、下一个任务块或文件结束终止。

若省略 `from self`，任务内 Plot 默认仍使用 `self`。

5.6.2.2 顶层 `@plot ... @end`

项目级绘图可引用多个任务或一组 matrix 变体：

```
@plot barrier rxn
  point r  label "R"  value gibbs_corr(freq_R) + single_point(sp_R)
  point ts label "TS" value gibbs_corr(freq_TS) + single_point(sp_TS)
  point p  label "P"  value gibbs_corr(freq_P) + single_point(sp_P)

  relative to r
  unit kcal/mol
  output "Results/Plots/rxn.png"
@end
```

顶层 `@plot` 必须以 `@end` 结束。图 ID 与图名相同，因此项目级图名必须唯一。

Plot 表达式同样按逻辑语句解析：未闭合圆括号和行尾逗号或算术运算符会继续读取下一物理行，函数调用允许尾随逗号。字符串和 snapshot 引用不能跨物理行。`series ...:`、`axis ...:` 与 `legend:` 的缩进体只在显式冒号后生效。

5.6.3 通用语句

所有图类型可使用以下通用语句：

```
from self
from task <task-name>
from variants <template-task>
from matrix <template-task>      # variants 的兼容别名
from derived("<derive-id>.<table-name>")
from derived("<derive-id>", "<table-name>")
from derived("<derive-id>", table="<table-name>")
from derived <derive-id>.<table-name>
from project

title "..."
output "Results/Plots/name.png"
format png|svg|pdf|dumb|cdxml
size <width> <height>
style lines|line|linepoints|linespoints|points|boxes|bars
theme auto|scientific|report|matrix|minimal|level
missing error|skip|nan

axis x|y|y2:
  label "..."
  unit "..."
  scale linear|log10

legend:
```

```
  position auto|top right|top left|bottom right|bottom left|outside
↪  right|bottom
  visible true|false
  order <series-id> ...
```

`theme` 控制坐标框、网格、图例和刻度的默认布局，不改变数据、单位或拟合语义。可用值及别名如下：

规范值	别名	用途
`auto`	`default`	按图类型选择主题。
`scientific`	`sci`、`classic`	科学曲线风格，默认无网格并保留图例和次刻度。
`report`	`dashboard`	报告风格，使用 y 网格和外向刻度。
`matrix`	`heatmap`	矩阵风格，隐藏图例并使用轻量边框。
`minimal`	`clean`	简洁风格，无网格和次刻度。
`level`	`levels`	能级图风格，隐藏图例并保留 y 网格。

`theme auto` 的映射为：`curve`、`spectrum`、`scatter`、`errorbar` 使用 `scientific`；`heatmap` 使用 `matrix`；`frontier`、`excited` 使用 `level`；`barrier`、`convergence`、`histogram`、`bar` 使用 `report`。

`terminal <format>` 也可接受；任务文件建议使用 `format` 和 `size`：

```
format png
size 1200 800
```

`output` 后缀为 `.png`、`.svg`、`.pdf`、`.dumb` 或 `.cdxml` 时，未显式写 `format` 会自动推导格式；若显式格式与后缀冲突，解析阶段直接报错。`format chemdraw` 是 `format cdxml` 的输入别名，格式化输出时统一写为 `cdxml`。

`cdxml` 适用于 `barrier`、`frontier` / `fmo` 与 `excited` / `excited-state`。其他图类型声明 CDXML 输出会在解析阶段报错。CDXML 由 BaneTask 原生写出，不调用 gnuplot；`size <width> <height>` 控制 ChemDraw 文档画布的坐标范围，省略时使用对应图类型的自动尺寸策略。

默认输出路径：

- 任务内图：`Results/Plots/[taskname]_<plot-name>.png`
- 项目级图：`Results/Plots/<plot-name>.png`

5.6.3.1 读取 Derive DSL 产物

`@plot` 可以直接把 `@derive` 产出的 table 作为数据源。推荐把科学筛选、单位换算、relative energy、Boltzmann population、分组统计等分析逻辑放在 `@derive`，再由 `@plot` 只声明列角色和渲染方式：

```
@derive benchmark
  from tasks("case_*")
  select task.name
  select reference = kv(self, "E_ref", unit=kcal/mol)
```

```
  select computed = energy(self, unit=kcal/mol)
  select error = computed - reference
  emit table points
@end

@plot scatter benchmark_corr
  from derived("benchmark.points")
  x reference label "Reference" unit kcal/mol
  y computed  label "Computed"  unit kcal/mol
  diagonal y=x
  fit linear
  metrics rmse mae r2
  output "Results/Plots/benchmark_corr.svg"
@end
```

除 table 数据源外，Plot 表达式可以直接读取 `.derived.json` 中的数值标量：

```
@plot barrier activation_profile
  point r  label "R"  value derived("thermo.G_R")
  point ts label "TS" value derived("thermo", "G_TS")
  point p  label "P"  value derived(derive="thermo", value="G_P")
  relative to r
  axis y:
    label "Relative Gibbs energy"
    unit "kcal/mol"
  output "Results/Plots/activation_profile.svg"
@end
```

`derive_value(...)` 是等价别名。标量 selector 支持 `derived("derive-id.value")`、两个位置参数，或 `derive=` / `value=` 具名参数；不能混用位置和具名形式。目标必须是 finite numeric scalar，文本、序列和表不能作为 plot 标量。selector 组件只接受标识符式名称，不能包含路径分隔或 `..` 路径跳转。

Plot artifact 的 `manifest.json` 会记录 `derive_dependencies` 和上游 derived fingerprint；derived JSON 内容变化时，plot render fingerprint 也会变化，从而触发重画。

在普通完整模式中，BaneTask 会先执行 Derive pipeline，再执行 Plot pipeline，然后物化隐式 finalize。因此上例中的 `benchmark.points` 可以由同一个任务文件里的 `@derive benchmark` 在本轮先生成，再被 `@plot scatter benchmark_corr` 读取；后续收尾动作也能读取本轮生成的派生与绘图结果。若上游任务结果尚未产生，导致本轮 `@derive` 被跳过且 `.derived.json` 尚不存在，读取该 derived table 的 plot 会作为依赖未完成而跳过，不把缺失 derived 文件记为 plot 失败。`--plots-only` 只运行 Plot pipeline，不会补跑上游 derive。

derived table 列可以通过标识符或字符串字面量引用：

```
x dG
y "mean error"
```

`heatmap` 与 `bar` 的 x / y 类别轴允许读取 string 列，并自动映射为稳定的类别位置。`curve`、`scatter`、`spectrum`、`histogram`、`errorbar` 的数值列必须能转换为 number。

5.6.3.2 命名 series

`curve` 和 `convergence` 可为序列声明稳定 ID。单行形式：

```
series dft = dft_energy label "DFT" style linepoints
series correlated = correlated_energy label "DLPNO-CCSD(T)" axis y2 style points
```

块形式适合较多修饰项：

```
series correlated = correlated_energy:
  label "DLPNO-CCSD(T)"
  unit "kcal/mol"
  axis y2
  style points
  scale log10
```

series ID 必须匹配 `[A-Za-z_][A-Za-z0-9_]*`，用于 legend 排序、manifest 和稳定的渲染身份；显示文本由 `label` 决定。可用 style 为 `lines`、`linepoints`、`points` 和 `boxes`。这些是后端无关的语义样式，不暴露 gnuplot pointtype/dashtype 数字。标准 convergence 能量与四类收敛准则仍保留专用渲染角色；命名 style 主要作用于 curve 和非标准 convergence 序列。

5.6.3.3 显式 axis

```
axis x:
  label "Dihedral angle"
  unit "degree"
  scale linear

axis y:
  label "Relative energy"
  unit "kcal/mol"

axis y2:
  label "Oscillator strength"
  scale log10
```

`axis unit` 是实际转换契约，不只是标签。x 坐标或挂载到同一显式 y/y2 轴的 series 必须具有可兼容维度，数值会换算到轴单位；不兼容单位或无可识别维度会在渲染前失败。`scale log10` 要求该轴上的所有可见 series 和 threshold 均为正。series 上的 `log` / `scale log10` 也按物理轴语义处理：同一轴的其他 series 会一起接受正值检查，不能依赖 renderer 静默丢弃非正点。每个 x、y、y2 轴最多声明一次，`label`、`unit` 和 `scale` 也不能在同一轴块重复。

5.6.3.4 Legend

```
legend:
  position outside right
  visible true
  order correlated dft
```

`order` 引用命名 series ID；未列出的 series 按原声明顺序追加，未知 ID 直接报错。可用位置为 `auto`、`top right`、`top left`、`bottom right`、`bottom left`、`outside right` 和

bottom。legend 块内的 **position**、**visible**、**order** 均最多声明一次。规范化输出会显式写出 **visible true|false**，保证默认配置仍可 parse-format-parse round trip。

5.6.3.5 缺失值策略

```
missing error    # 默认；任一样本缺值即报错
missing skip     # 丢弃该完整样本，x 与全部 y 同步删除
missing nan      # 保留样本并向 gnuplot 写入 NaN
```

missing skip 以样本为事务单位，不会保留只写入了 x 或部分 y 的半成品行。

5.6.4 表达式

表达式支持：

- 数字；
- 单引号或双引号字符串；
- 标识符；
- 函数调用和具名参数；
- **+**、**-**、*****、**/**；
- 一元正负号；
- 括号；
- 显式轻量 snapshot 引用 **{{ task.key }}**。

示例：

```
value gibbs_corr(freq_TS) + single_point(sp_TS)
value energy(ts, kind=correlated) - energy(r, kind=correlated)
value 627.509474 * (energy(ts) - energy(r))
value kv(ts, "shermo_g", unit=hartree)
value {{ ts.shermo_g }}
```

任务名包含连字符时建议加引号：

```
value energy("scan-010")
```

5.6.4.1 规范结果函数

以下函数读取规范结果数据，不依赖轻量快照 key 猜测：

表达式	含义	典型形状
energy(task)	最终 step 的 primary energy	scalar
energy(task, kind=scf)	最终 SCF energy	scalar
energy(task, kind=correlated)	最终相关能 / post-HF 能量	scalar
energy(task, kind=double_hybrid) / **energy(task, kind=dh)**	最终 double-hybrid DFT 总能量	scalar
energy(task, kind=composite)	最终 composite method 能量	scalar

表达式	含义	典型形状
`energy(task, kind=composite_zero_k)`	最终 0 K composite method 能量	scalar
`energy(task, kind=gibbs)`	最终 Gibbs free energy	scalar
`single_point(task)`	`energy(task)` 的科研语义别名	scalar
`scf_energy(task)`	最终 SCF energy	scalar
`correlated_energy(task)`	最终相关能 / post-HF 能量	scalar
`double_hybrid_energy(task)` / `dh_energy(task)`	最终 double-hybrid DFT 总能量	scalar
`composite_energy(task)` / `composite_zero_k_energy(task)`	最终 composite / 0 K composite 能量	scalar
`free_energy(task)` / `gibbs_energy(task)`	Gibbs free energy	scalar
`gibbs_corr(task)`	Gibbs correction	scalar
`zpe(task)`	零点能或零点修正	scalar
`thermal_corr(task)`	热修正	scalar
`nimag(task)`	虚频数	scalar
`step.ordinal`	优化步序号	series
`step.energy`	优化步主能量；在 `convergence` 图中会额外包含紧跟优化后的 frequency Link1 检查步	series
`step.convergence(max_force)`	最大力序列；`convergence` 图中可包含 post-opt frequency 检查步	series
`step.convergence(rms_force)`	RMS 力序列	series
`step.convergence(max_displacement)`	最大位移序列	series
`step.convergence(rms_displacement)`	RMS 位移序列	series
`vibration.frequency`	振动频率	series
`vibration.ir_intensity`	IR 强度	series

兼容短名包括：

```
opt.step
opt.energy
opt.max_force
opt.rms_force
opt.max_displacement
opt.rms_displacement
```

`result(task, property)` 可通过稳定属性名查询同一结果访问层：

```
value result(ts, "qc.energy.scf")
y result(self, "qc.optimization.max_force")
```

属性列表覆盖 energy、thermochemistry、optimization、vibration、excited-state 和 orbital 属性，包括标量与带语义轴的序列。可用以下命令查看完整列表：

```
btproc result list-properties
btproc result describe-property qc.vibration.frequency
```

例如：

```
qc.energy.primary
qc.energy.scf
qc.energy.correlated
qc.energy.double_hybrid
qc.energy.composite
qc.energy.composite_zero_k
qc.thermochemistry.gibbs_energy
qc.thermochemistry.gibbs_correction
qc.thermochemistry.entropy
qc.thermochemistry.cp
qc.thermochemistry.temperature
qc.optimization.energy
qc.optimization.max_force
qc.vibration.frequency
qc.vibration.reduced_mass
qc.vibration.ir_intensity
qc.excited_state.wavelength
qc.excited_state.oscillator_strength
```

属性描述包含 shape、物理维度、规范单位、序列轴和解析需求。完整属性和状态语义见“统一量化结果查询”章节。

5.6.4.2 显式 snapshot 替代值

轻量 `.bane.result.kv` / snapshot 数据不会作为 `energy()` 等规范函数的隐式 fallback。需要访问时必须明确写：

```
value kv(task_name, "custom_key")
value kv(task_name, "custom_key", unit=hartree)
value {{ task_name.custom_key }}
```

`kv()` 只接受可转换为数字的值。给出 `unit=` 时，该单位会成为数值的输入单位，便于后续能量换算。

5.6.4.3 `self` 与 matrix 变量

任务内图中，`self` 指当前任务。`from variants scan` 的 curve 中，表达式会对每个展开任务求值，`self` 指当前变体：

```
@plot curve torsion
  from variants scan
  x matrix.dihedral label "Dihedral" unit degree
  y energy(self) label "Relative energy" unit kcal/mol
  relative minimum
  sort by x
@end
```

`matrix.<axis>` 从展开任务记录的 matrix 值读取。`from matrix scan` 与 `from variants scan` 等价，但推荐新语法使用 `variants`，因为被选择的是任务模板的展开变体，而不是一个外部矩阵文件。

5.6.5 图类型

5.6.5.1 barrier

语法：

```
@plot barrier <name>
  point <id> label "..." value <scalar-expression>
  ...
  relative to <point-id>
  unit hartree|eV|kcal/mol|kJ/mol|cm-1
@end
```

`point` 也可写成多行：

```
point ts
  label "TS"
  value gibbs_corr(freq_TS) + single_point(sp_TS)
```

完整示例：

```
@plot barrier rxn_profile
  point r  label "R"  value gibbs_corr(freq_R) + single_point(sp_R)
  point ts label "TS" value gibbs_corr(freq_TS) + single_point(sp_TS)
  point p  label "P"  value gibbs_corr(freq_P) + single_point(sp_P)

  relative to r
  unit kcal/mol
  title "Reaction profile"
  output "Results/Plots/rxn_profile.svg"
  format svg
  size 1000 700
@end
```

所有 point 必须求得能量维度标量。内部先统一到基础能量单位，再执行相对值变换和输出单位转换。

输出为 `.cdxml` 时，每个反应路径点绘制为独立的水平能级线，相邻点之间使用具有水平切线的 ChemDraw 贝塞尔曲线连接。能量数值、状态标签、标题、坐标轴和刻度均为独立可编辑对象；打开文件后可在 ChemDraw 中移动标签、调整曲线和线宽，或添加分子结构。省略

`size` 时，画布宽度会随 point 数量自动增加。

```
@plot barrier rxn_profile_chemdraw
  point r  label "R"  value gibbs_corr(freq_R) + single_point(sp_R)
  point ts label "TS" value gibbs_corr(freq_TS) + single_point(sp_TS)
  point p  label "P"  value gibbs_corr(freq_P) + single_point(sp_P)
  relative to r
  unit kcal/mol
  title "Reaction profile"
  output "Results/Plots/rxn_profile.cdxml"
@end
```

5.6.5.2 convergence

`convergence` 要求一个 x 序列和至少一个 y 序列。项目级图必须显式指定 `from task`；任务内图默认 `from self`。

```
$opt_TS
  %plot convergence conv
    x step.ordinal label "Step"
    y step.energy label "Energy" unit hartree
    y step.convergence(max_force) label "Max force"
    y step.convergence(rms_force) label "RMS force"
    y step.convergence(max_displacement) label "Max displacement"
    y step.convergence(rms_displacement) label "RMS displacement"
    # optional threshold setting
    threshold step.convergence(max_force) value 4.5e-4 label "Max force limit"
    threshold step.convergence(rms_force) value 3.0e-4 label "RMS force limit"
    threshold step.convergence(max_displacement) value 1.8e-3 label "Max
     ↪  displacement limit"
    threshold step.convergence(rms_displacement) value 1.2e-3 label "RMS
     ↪  displacement limit"

    style linepoints
    output "Results/Plots/[taskname]_conv.png"
```

`convergence` 图对标准收敛准则采用专用绘图策略：`max_force`、`rms_force`、`max_displacement` 和 `rms_displacement` 共用左侧对数坐标轴，不同准则使用不同颜色；当 `step.convergence(...)` 序列携带原生收敛阈值时，图中会自动绘制对应虚线阈值并标注标签，不需要额外写 `threshold` 行。同一准则若已显式声明 `threshold`，则使用显式声明。低于阈值的已收敛点使用浅色显示，高于阈值的点使用常规颜色突出。能量序列会自动绘制到右侧 y2 轴。非标准序列仍按普通 PlotDSL 序列绘制。

`axis y2` 把非标准序列或非标准阈值放到右轴；`log` 对对应轴启用对数刻度。对数序列中出现零或负数会在渲染前报错。

`convergence` 图在读取 `step.ordinal`、`step.energy` 和 `step.convergence(...)` 时，会把优化 job 之后紧跟的 frequency Link1 检查步作为末尾检查点纳入图中。这样 `opt freq` 任务可以显示优化收敛后进入频率计算时的梯度/收敛状态；普通 `curve` 图直接查询 `step.energy` 时仍只使用优化历史。

5.6.5.3 curve

`curve` 支持普通单任务序列，也支持 matrix 变体聚合。

优化能量历史：

```
@plot curve opt_energy
  from task opt_TS
  x step.ordinal label "Step"
  y step.energy label "Energy" unit hartree
  style linepoints
@end
```

matrix scan：

```
@plot curve torsion_scan
  from variants scan
  x matrix.dihedral label "Dihedral angle" unit degree
  y energy(self) label "Relative energy" unit kcal/mol
  relative minimum
  sort by x
  missing error
  output "Results/Plots/torsion_scan.pdf"
@end
```

Gaussian Huang–Rhys 因子对频率示例：

```
@plot bar huang_rhys
  from task fcht
  x result(fcht, "qc.vibration.frequency") label "Frequency" unit cm-1
  y result(fcht, "qc.vibronic.huang_rhys") label "Huang-Rhys factor"
  output "Results/Plots/huang_rhys.svg"
@end
```

这两个属性都以 Gaussian 振动模态编号为 key，Plot DSL 会按 key 对齐后作图；因此横轴使用实际频率，而不是把模态编号误作等距类别。

支持：

```
relative first
relative minimum
relative none
sort by x
```

`curve` 不支持 `relative to <id>`；该语法只属于 barrier point。

命名 series、双轴和 legend 示例：

```
@plot curve method_comparison
  from derived("scan.points")
  x angle
  series dft = dft:
    label "DFT"
    style linepoints
  series correlated = correlated:
```

```
    label "DLPNO-CCSD(T)"
    axis y2
    style points
  axis x:
    label "Angle"
    unit "radian"
  axis y:
    label "DFT energy"
    unit "kcal/mol"
  axis y2:
    label "Correlated energy"
    unit "cm-1"
  legend:
    position top left
    order correlated dft
  output "Results/Plots/method_comparison.svg"
@end
```

5.6.5.4 spectrum

spectrum 用于把离散谱线画成 stick spectrum、Gaussian broadened spectrum，或两者叠加。输入可以来自任务结果序列，也可以来自 derived table。对于已经由量化程序采样好的连续谱，可使用 **mode sampled** 原样绘制，不再做二次展宽。

IR 示例：

```
@plot spectrum ir
  from task freq
  x vibration.frequency label "Frequency" unit cm-1
  intensity vibration.ir_intensity label "IR intensity" unit km/mol
  mode sticks+broadened
  broaden gaussian fwhm 12 unit cm-1
  range 400 4000
  output "Results/Plots/ir.svg"
@end
```

derived table 示例：

```
@plot spectrum uvvis
  from derived("td.lines")
  x wavelength label "Wavelength" unit nm
  intensity oscillator_strength label "Oscillator strength"
  mode sticks+broadened
  broaden gaussian fwhm 0.30 unit eV
  output "Results/Plots/uvvis.png"
@end
```

Gaussian FC / HT / FCHT 振动分辨光谱示例：

```
@plot spectrum vibronic
  from task fcht
  x result(fcht, "qc.vibronic.spectrum.energy") label "Wavelength" unit nm
```

```
  intensity result(fcht, "qc.vibronic.spectrum.intensity") label "Molar
  ↪  absorption coefficient"
  mode sampled
  sort by x
  output "Results/Plots/vibronic.svg"
@end
```

`spectrum` 的 x 轴按光谱坐标处理。绝对光谱坐标可在 `cm-1`、`eV`、`nm` 和 `hartree` 之间转换；因此上例也可将 x 轴写成 `label "Wavenumber" unit cm-1` 或 `label "Photon energy" unit eV`。波数转换为波长后采样点通常不再等间距，并且坐标方向会反转，使用 `sort by x` 可按转换后的坐标重新排序。IR 的 `qc.vibration.frequency` 在 `spectrum` x 轴位置也使用同一套光谱坐标转换。

这里 `qc.vibronic.spectrum.intensity` 优先选择与 Gaussian 模拟温度最接近的 `Final Spectrum` 强度列；要固定使用 0 K 列，可改为 `qc.vibronic.spectrum.intensity_0k`。Gaussian 输出中的物理量名称和单位会分开解析，结果单位只包含如 `dm^3.mol^-1.cm^-1` 的单位文本，不会把 `Molar absorption coefficient` 再次拼进坐标轴单位。`sampled`（别名 `native`、`grid`）把 x / intensity 当作量化程序已经采样好的曲线，允许 `range` 和 `sort by x`，但不能再声明 `broaden`。

支持：

```
mode
↪  sticks|stick|broadened|smooth|gaussian|sticks+broadened|stick+broadened|both|overlay|
broaden gaussian fwhm <number> [unit <unit>]
range <min> <max>
sort by x
```

`sticks+broadened`、`stick+broadened`、`both` 和 `overlay` 等价，都会同时绘制离散谱线和展宽曲线。

FWHM 是宽度而不是绝对光谱坐标。`cm-1`、`eV` 与 `hartree` 之间可以线性换算；涉及 `nm` 时宽度换算依赖谱带中心，因此 x 轴使用 `nm` 时应直接以 `nm` 给出 FWHM。

5.6.5.5 `scatter` / `correlation`

`scatter` 用于 computed vs experimental、method A vs method B、benchmark 相关图等。`correlation` 是 `scatter` 的别名。

```
@plot correlation method_compare
  from derived("benchmark.points")
  x E_ref  label "Reference" unit kcal/mol
  y E_test label "Test"      unit kcal/mol
  diagonal y=x
  fit linear
  metrics r2 rmse mae
  output "Results/Plots/method_compare.pdf"
@end
```

支持：

```
diagonal
diagonal y=x
fit linear|none
metrics rmse mae r2
```

`diagonal`、`diagonal y=x`、`diagonal yx` 与 `diagonal true` 等价。

打开 `fit linear` 时，数据集中会加入一条线性拟合线。`metrics` 会计算并写入 artifact manifest；当前渲染器也会把 metrics 拼入图标题。

5.6.5.6 heatmap

`heatmap` 使用 x、y、z 三列生成二维色图。x / y 可以是数值列，也可以是 derived table 中的 string 类别列。

二维 PES 示例：

```
@plot heatmap pes2d
  from derived("pes.grid")
  x phi label "Phi" unit degree
  y psi label "Psi" unit degree
  z dE  label "Relative energy" unit kcal/mol
  output "Results/Plots/pes2d.svg"
@end
```

method × basis 示例：

```
@plot heatmap method_basis
  from derived("benchmark.summary")
  x method
  y basis
  z MAE unit kcal/mol
@end
```

Gaussian Duschinsky 矩阵示例：

```
@plot heatmap duschinsky
  from task fcht
  x result(fcht, "qc.vibronic.duschinsky.column") label "Final-state mode"
  y result(fcht, "qc.vibronic.duschinsky.row") label "Initial-state mode"
  z result(fcht, "qc.vibronic.duschinsky.matrix") label "Duschinsky J"
  theme matrix
  output "Results/Plots/duschinsky.svg"
@end
```

三个 Duschinsky 属性共享 `duschinsky_element` 轴，行、列均为从 1 开始的数值索引。

ORCA 旋轨耦合矩阵元示例：

```
@plot heatmap soc_matrix
  from task soc
  x result(soc, "qc.spin_orbit_coupling.singlet_index") label "Singlet state"
  y result(soc, "qc.spin_orbit_coupling.triplet_index") label "Triplet state"
  z result(soc, "qc.spin_orbit_coupling.magnitude") label "|SOC|" unit cm-1
```

```
  theme matrix
  output "Results/Plots/soc_matrix.svg"
@end
```

三条 SOC 序列共享 `spin_orbit_coupling_element` 轴和相同 key；x/y 使用实际态编号，包括输出中存在的 `S0` 编号 0。`z` 是三个复分量平方模之和的平方根。矩阵默认采用 `M_S` 优先、XYZ 回退策略；因此同一态对不会因两套 ORCA 输出而重复成两个热图单元格。

数值型 heatmap 会按网格间距自动扩展半个单元格，例如模态 `1..N` 的显示范围为 `0.5..N+0.5`，避免首末单元格被裁掉。渲染器会生成三列 `data.tsv`：`x`、`y`、`z`，按 y 方向扫描组织为规则网格，并用 `NaN` 补齐稀疏矩阵中缺失的组合。热图默认使用白、橙、红渐变色带，色层会铺满整个网格单元，并在每个具有有效 z 值的色块中央以两位小数标注数值；缺失值不绘制色块和数字。

5.6.5.7 `histogram` / `hist`

`histogram` 用于绘制数值分布，例如 conformer ΔG、benchmark error、charge distribution。

```
@plot histogram conf_dg_hist
  from derived("conformer_inventory.conformers")
  x dG label "Delta G" unit kcal/mol
  bins 30
  range 0 10
  density false
  output "Results/Plots/conf_dg_hist.png"
@end
```

支持：

```
bins <positive-integer>
range <min> <max>
density true|false
```

`density true` 时，y 值为归一化密度；默认 y 值为 count。

5.6.5.8 `bar` / `barchart`

`bar` 用于类别柱图或数值横轴柱图，例如 conformer population、fragment contribution、EDA component summary，以及按振动频率排列的 Huang–Rhys 因子。x 可以是 string 类别列，也可以是数值序列；数值横轴会保留真实 x 坐标，并根据相邻点间距自动设置柱宽。

```
@plot bar population
  from derived("conformer_inventory.conformers")
  x name
  y population label "Population"
  sort by y desc
  limit 20
  output "Results/Plots/conformer_population.svg"
@end
```

支持：

```
sort by y asc|ascending|desc|descending
sort none
limit <positive-integer>
```

5.6.5.9 errorbar / errorbars

errorbar 用于 group mean ± standard error / standard deviation 等统计图。

```
@plot errorbar grouped_error
  from derived("benchmark.grouped")
  x method_index label "Method"
  y mean_error label "Mean error" unit kcal/mol
  yerr std_error unit kcal/mol
  sort by x
  output "Results/Plots/grouped_error.svg"
@end
```

yerr 与 **y** 维度相同时会自动转到 y 的输出单位；显式写 **yerr ... unit <unit>** 可覆盖错误条输入单位。

5.6.5.10 frontier / fmo

frontier 用于比较一个或多个体系的前线轨道能级。渲染器用水平短线表示轨道能级，用箭头表示占据电子；闭壳层轨道在占据数接近 2 时绘制成一对上下箭头，开壳层 alpha / beta 轨道可分列显示。**fmo** 是 **frontier** 的别名。

项目级比较示例：

```
@plot frontier fmo_compare
  column donor    label "Donor"    task donor_sp
  column acceptor label "Acceptor" task acceptor_sp
  orbitals HOMO-1 HOMO LUMO LUMO+1
  spin both
  unit eV
  output "Results/Plots/fmo_compare.svg"
@end
```

任务内示例：

```
$radical_sp
  %plot frontier fmo
    from self
    orbitals HOMO LUMO
    spin auto
    unit eV
    output "Results/Plots/[taskname]_fmo.svg"
```

支持的专用语句：

```
column <id> [label "..."] [task <task-name>]
system <id> [label "..."] [task <task-name>]    # column 的别名
orbitals HOMO-1 HOMO LUMO LUMO+1
orbital HOMO LUMO                               # orbitals 的别名
```

```
spin auto|alpha|beta|both
unit hartree|eV|kcal/mol|kJ/mol|cm-1
```

`column` 用于显式指定参与比较的体系。省略 `task` 时，任务名默认等于 `<id>`；省略 `label` 时，显示标签默认等于 `<id>`。未写 `column` 时，任务内图默认读取 `from self`，项目级图可使用 `from task <task-name>` 或 `from variants <template-task>`。

`orbitals` 控制纳入图中的轨道。常用选择器包括 `HOMO`、`HOMO-1`、`LUMO`、`LUMO+1`；也可以直接写正整数轨道序号。若省略该语句，默认绘制 `HOMO-1 HOMO LUMO LUMO+1`。

`spin auto` 读取默认 / alpha 轨道通道，并根据占据数判断闭壳层或单占据轨道；`spin alpha` 只读取 alpha 通道，`spin beta` 只读取 beta 通道，`spin both` 同时读取 alpha 与 beta。对于只提供一套闭壳层轨道的结果，`spin both` 会保留默认 / alpha 通道并绘制成成对电子。

`frontier` 需要任务结果中存在轨道能量、占据数和必要的自旋标记。解析阶段会为相关任务声明 `orbitals` 结果需求。渲染时默认在每条轨道能级旁标注轨道名和能量数值，并使用固定的横线目标长度；未显式指定 `output size` 时，画布宽高会随比较列数和每列轨道数量自动调整。生成的 `data.tsv` 包含 `column`、`label`、`task`、`spin`、`orbital`、`index`、`energy`、`occupation` 等列。

输出为 `.cdxml` 时，每条能级横线、alpha / beta 电子箭头、轨道标签、能量数值、坐标轴和刻度均写成独立 ChemDraw 对象，可在 ChemDraw 中移动、改字、改线宽或与分子结构组合。闭壳层双占据轨道保留一对方向相反的电子箭头；同时存在 alpha 与 beta 通道时，两套轨道在同一体系列内左右分开，并分别加 `α` / `β` 标签。

```
@plot frontier fmo_chemdraw
  column donor label "Donor" task donor_sp
  column acceptor label "Acceptor" task acceptor_sp
  orbitals HOMO-1 HOMO LUMO LUMO+1
  spin both
  output "Results/Plots/fmo_chemdraw.cdxml"
@end
```

5.6.5.11 `excited` / `excited-state`

`excited` 用于比较一个或多个任务的激发态能级。渲染器用水平短线表示每个 root 的激发态能量，默认在能级旁标注态标签和能量数值。态标签优先读取激发态自旋多重度：单重态、双重态、三重态、四重态分别标注为 `S<n>`、`D<n>`、`T<n>`、`Q<n>`；结果中未报告多重度时使用 `S<n>`。`excited-state`、`excited_state`、`excited-level` 等写法均作为别名接受。

项目级比较示例：

```
@plot excited es_compare
  column td_a label "TD-A" task td_a
  column td_b label "TD-B" task td_b
  roots 1 2 3
  arrow from td_a root 1 to td_b root 2 style solid
  arrow from td_a root 2 to td_b first lower style dashed
  arrow from td_a root 3 to td_b first higher dashed
  unit eV
  output "Results/Plots/es_compare.svg"
@end
```

任务内示例：

```
$td
  %plot excited es
    from self
    roots 1 2 3
    unit eV
    output "Results/Plots/[taskname]_excited.svg"
```

支持的专用语句：

```
column <id> [label "..."] [task <task-name>] [multiplicity
 ↪  singlet|doublet|triplet|quartet|< 正整数 >]
system <id> [label "..."] [task <task-name>] [multiplicity ...]    # column 的
 ↪  别名
roots 1 2 3
roots all                                          # 省略 roots 或写 all 均表示读取全部可
 ↪  用 root
arrow from <column> root <n> to <column> root <m> [style
 ↪  solid|line|dashed|dash|dotted]
arrow from <column> root <n> to <column> first lower|below  [style
 ↪  solid|line|dashed|dash|dotted]
arrow from <column> root <n> to <column> first higher|above [style
 ↪  solid|line|dashed|dash|dotted]
unit hartree|eV|kcal/mol|kJ/mol|cm-1
```

column 的含义与 **frontier** 相同：用于显式指定参与比较的任务列，省略 **task** 时任务名默认等于 **<id>**。未写 **column** 时，任务内图默认读取 **from self**，项目级图可使用 **from task <task-name>** 或 **from variants <template-task>**。

multiplicity 用于把同一任务中的不同自旋多重度拆成独立列。可写 **singlet** / **S**、**doublet** / **D**、**triplet** / **T**、**quartet** / **Q**，也可直接写正整数。指定后，**roots** 与箭头中的 root 按该多重度内部的能量结果顺序从 1 重新编号；例如 ORCA 在同一 SOC 计算中把三重态打印为全局 **STATE 7** 至 **STATE 12**，三重态列中仍按 **T1** 至 **T6** 使用。

```
@plot excited es_compare
  column S label "Singlet TD" task soc multiplicity singlet
  column T label "Triplet TD" task soc multiplicity triplet
  roots 1 2 3 4 5 6
  arrow from S root 1 to T root 1 style solid
  unit eV
  output "Results/Plots/soc_excited_compare.svg"
@end
```

roots 控制基础可见激发态 root。若省略 **roots** 或写 **roots all** / **roots ***，会绘制结果中全部可用 root；显式写 **roots 1 2 3** 时，初始只绘制指定 root。箭头端点可以位于 **roots** 之外：显式箭头会自动补画被引用的端点 root；相对箭头若在目标列命中 **roots** 之外的 root，会自动补画目标列中从低 root 到该 root 的所有态，避免出现 **1 2 3 5** 这种断层显示。

显式箭头使用 **arrow from A root 1 to B root 2**。相对目标使用 **first lower** 或 **first higher**：例如 **arrow from A root 2 to B first lower style dashed** 会先读取 A 列 root 2 的能量，再在 B 列全部可用 root 中寻找能量最接近且低于它的 root 作为箭头终

点；`first higher` 则寻找能量最接近且高于它的 root。箭头样式默认是 `solid`。`line` 等价于 `solid`；`dash`、`dotted` 等价于 `dashed`。样式可以写在 `style` 之后，也可直接放在语句末尾。相对方向 `below` 等价于 `lower`，`above` 等价于 `higher`。

`excited` 读取统一结果属性 `qc.excited_state.excitation_energy`；若结果中存在 `qc.excited_state.multiplicity`，渲染器会用该属性决定 root 标签前缀。解析阶段会为相关任务声明 `excited-states` 结果需求。渲染时默认在每条能级旁标注态标签和能量，能级横线使用固定的目标长度；未显式指定 `output size` 时，画布宽高会随比较列数和每列 root 数量自动调整。生成的 `data.tsv` 包含能级行和箭头行；artifact manifest 会记录 `points` 与 `arrows` 数量。

输出为 `.cdxml` 时，激发态能级、态标签、能量数值和跃迁箭头均为可编辑 ChemDraw 对象。`style dashed` 会写成 ChemDraw 虚线箭头；同列跃迁放在能级横线左侧，不同列跃迁从源列横线外缘连到目标列横线外缘，箭头指向 `to` 端。

```
@plot excited es_chemdraw
  column singlet label "Singlet" task td_s
  column triplet label "Triplet" task td_t
  roots 1 2 3
  arrow from singlet root 1 to triplet first lower style dashed
  output "Results/Plots/es_chemdraw.cdxml"
@end
```

5.6.6 单位

当前会进行实际数值换算的能量单位：

```
hartree
 eV
kcal/mol
kJ/mol
cm-1
```

单位名称不区分大小写，并接受常见拼写归一化。能量声明还接受 `eh`、`au`、`a.u.`、`kcalmol-1`、`kjmol-1`；角度声明接受 `degree`、`radian`、`rad`；长度声明接受 `angstrom`、`bohr`；频率声明接受 `cm-1`、`cm^-1`、`1/cm`。换算顺序为：

```
输入结果 - 基础能量单位 - relative 变换 - 输出单位
```

显式 axis 会使用统一结果层的单位契约执行可用换算，包括能量、角度、长度、光谱坐标、熵、热容和偶极矩等已登记维度；例如 `degree` 可换算为 `radian`。频率与 spectral-coordinate 的 `cm-1` 语义按源结果 dimension 区分，不仅凭单位字符串猜测。未登记或不兼容单位直接报错。

表达式中的加减要求形状和维度兼容；乘除允许标量缩放。标量与 series 的运算会按元素广播。不能把字符串、矩阵或无关维度直接相加。

5.6.7 结果加载与需求分析

解析 Plot DSL 时，BaneTask 会提前推导每个任务所需的数据：

```
energy(task) / energy(task, kind=...) - final-energy
step.energy                 - energy-history
step.convergence(...)       - convergence-history
free_energy/gibbs_corr      - thermochemistry
vibration.frequency         - vibrations
frontier/fmo                - orbitals
excited/excited-state       - excited-states
derived(derive, value)       - derived numeric scalar and derive dependency
kv()/{{...}}                - legacy-kv
```

运行时加载顺序：

1. 当前任务目录或集中归档中的规范 **bane.qc.document.v2** JSON；
2. 若归档不存在或不满足数据需求，从任务输出日志解析一次；
3. 同一轮绘图中按任务缓存查询结果；
4. 只有 **kv()** / **{{...}}** 才访问轻量 snapshot。

候选规范归档包括任务目录中的 **.bane.qc.json**、**qc_document.json**、**raw_result.json**，以及 BaneTask 集中 artifact 中的 **raw_result.json**。

普通 **banetask** 完整流程中，Derive pipeline 会先于 Plot pipeline 执行，二者均先于隐式 finalize；尚未产生任务结果或 derived 结果的 plot 会被跳过，不阻断任务准备；**--plots-only** 则要求已有结果，缺失会导致失败。

5.6.8 Artifact 与可复现性

每张图会保存公开输出和独立 artifact 目录：

```
Results/Plots/rxn.png
Results/Artifacts/plots/rxn/
├── data.tsv
├── render.gp
├── spec.plotdsl
└── manifest.json

Results/Plots/fmo.cdxml
Results/Artifacts/plots/fmo/
├── data.tsv
├── render.cdxml
├── spec.plotdsl
└── manifest.json
```

任务内图 ID 中的 **:** 等字符会在 artifact 目录名中安全化。

manifest.json 使用 schema：

```
banetask.plot-artifact.v1
```

其中记录：

- plot ID、名称、类型和 scope；
- 输出文件、数据、渲染源和规范化 spec 路径；CDXML artifact 还记录 **document** 路径；
- **data_fingerprint** 与 **render_fingerprint**；

- renderer、格式和数据点数；
- 依赖任务、derive 依赖和上游 fingerprint；
- 每个任务所需的结果类别；
- scatter / correlation metrics；
- series 的标签、单位、轴和实际渲染 style。

BaneTask 始终保留 **data.tsv**、规范化 **spec.plotdsl** 和对应的渲染源。gnuplot 输出归档 **render.gp**；ChemDraw 输出归档 **render.cdxml**。

当输出文件存在且 render fingerprint 未变化时，默认跳过渲染。**--rerender-plots** 可强制重新渲染；对于 CDXML，它会重新运行原生写出器而不是调用 gnuplot。

5.6.9 CLI

```
banetask [options] file.bt
banetask [options] file.btplt
```

Plot 相关选项：

```
--no-plots              不分析或渲染 plot artifact
--plots-only            只使用已有结果渲染图，不准备任务
--plot <name-or-id>     只处理指定图；任务内 ID 形如 task:plot
--rerender-plots        忽略 fingerprint，强制渲染
--explain-plots         只显示依赖和数据需求，不读取结果或渲染
--gnuplot <path>        指定 PNG、SVG、PDF 与 dumb 输出使用的 gnuplot 可执行文件
```

示例：

```
banetask case.bt --explain-plots
banetask case.bt --plots-only
banetask case.bt --plots-only --plot rxn
banetask case.bt --plots-only --plot opt_TS:conv --rerender-plots
banetask case.bt --gnuplot /opt/gnuplot/bin/gnuplot
```

--explain-plots 示例输出：

```
plot rxn [barrier]
  scope: project
  output: Results/Plots/rxn.png
  dependencies: freq_R sp_R freq_TS sp_TS
  derive dependencies: none
  requirements:
    freq_R: thermochemistry
    sp_R: final-energy
```

配置文件或环境变量可设置：

```
BANETASK_GNUPLOT=gnuplot
```

命令行 **--gnuplot** 优先级最高。CDXML 输出不读取该选项，也不要求系统安装 gnuplot。

5.6.10 解析、include 与 matrix 展开

Plot block 会作为 DSL 块参与以下处理：

- `define:` 与任务变量替换；
- `[taskname]` 等占位符替换；
- `&INCLUDE`；
- canonical formatter round-trip；
- matrix 任务复制与任务引用重写；
- 任务依赖静态检查。

matrix 模板任务中的 `%plot` 会随任务展开，每个变体得到独立的 task-scoped plot 和默认输出名。例如：

```
$scan
  %plot convergence conv
    from self
    x step.ordinal
    y step.energy
```

展开后可形成：

```
scan__angle_0:conv
scan__angle_10:conv
...
```

项目级 `from variants scan` 则把这些变体聚合为同一条 curve 的样本。

5.6.11 常见诊断

Plot 声明在解析、依赖检查、结果查询、单位换算和渲染阶段都会给出诊断。常见问题包括：

- 未知 plot kind 或 statement；
- `barrier` 缺少 `point` / `value`；
- `convergence` 或 `curve` 缺少 `x` 或 `y`；
- 引用了不存在的任务，或模板任务没有展开变体；
- `frontier` 选择的轨道不存在，或任务结果不包含轨道数据；
- `excited` 选择的 root 不存在，或 `first lower` / `first higher` 找不到满足条件的目标 root；
- 属性名不存在；
- 标量和序列混用；
- 单位或物理维度不兼容；
- `log` 序列包含非正数；
- `output` 后缀与 `format` 冲突；
- 找不到 gnuplot，或 gnuplot 未生成目标文件。

排查时先运行生成命令确认依赖任务已有结果，再检查属性名、单位和输出路径。需要查看 matrix 展开后的实际 plot，可使用 `btool expand task.bt --expanded`。

5.7 独立 Plot 脚本

`.btplt` 用于把一个 Plot 声明从主 `.bt` 文件中拆出，形成可单独复制、双击或从命令行执行的绘图脚本。它只读取已有结果，不准备计算任务。

5.7.1 文件配对

显式打开 `.btplt` 时，BaneTask 只在脚本所在目录的当前层查找 `.bt`：

```
case/
  case.bt
  correlation.btplt
```

目录必须恰好包含一个 `.bt` 文件；不递归检查子目录，`.projbt`、`.btfrag` 和 `.btlib` 不参与计数。配对 `.bt` 提供任务、变量、matrix 展开和 case 根目录，绘图结果仍写入该 case 的 `Results/`。每个独立脚本只允许声明一个 plot。

5.7.2 文件格式

第一条非空、非注释行写 `<plot-kind> [name]`，后续直接写 `@plot` 块内部语句：

```
# correlation.btplt
scatter benchmark_corr
from derived("benchmark.table")
x reference label "Reference" unit kcal/mol
y predicted label "Predicted" unit kcal/mol
fit linear
reference y=x
output "Results/Plots/benchmark_corr.svg"
```

名称可省略，此时文件名主体作为 plot name 和 id，默认输出路径也使用该名称：

```
# correlation.btplt
scatter
from task sp
x energy(sp) label "Reference"
y energy(sp) label "Predicted"
output "Results/Plots/correlation.svg"
```

也可保留完整 `@plot ... @end` 包装；独立文件末尾的 `@end` 可省略。可用图类型与 Plot DSL 相同。

5.7.3 执行与限制

```
banetask correlation.btplt
banetask correlation.btplt --explain-plots
banetask correlation.btplt --rerender-plots
banetask correlation.btplt --plot correlation
banetask correlation.btplt --gnuplot C:\tools\gnuplot\bin\gnuplot.exe
```

该入口使用 plots-only 路径，并把本次执行视图中的 plot 集合替换为当前脚本声明；配对 `.bt` 中其他 plot 不会一并渲染，磁盘文件不修改。`.btplt` 不会自动运行上游 derive；使用 `from derived(...)` 或标量 `derived(...)` 时，对应 `.derived.json` 必须已存在。`--plot` 只能选择当前脚本名称或 id；`.btplt` 不能与 derive-only、derive explain、derive selector 或 derive rerender 选项混用。

当前一个 plot 仍对应一个 source 数据模型和一个 output。多 source layer、多格式 `export`、reaction-network barrier 以及多面板 figure 尚未进入该语法；需要数据连接、分组统计或拟合推断时，应先在 Derive DSL 中生成带 provenance 的表或标量，再由 Plot DSL 呈现。

Windows 安装器会把 `.btplt` 的打开命令关联到 `banetask.exe "%1"`。分发时应说明脚本依赖的任务、derived id、结果属性和 renderer，并把脚本放入恰好包含一个 `.bt` 的 case 目录。

6 附录

本章集中列出文件类型、声明、任务指令、展开阶段、配置、生成文件、状态、CLI 和规范结果属性。正式行为仍由前述章节的同名条目定义。

6.1 文件类型

后缀或名称	作用	执行入口
`.bt`	单 case 任务、可含项目级 Derive/Plot	`banetask file.bt`
`.projbt`	project 入口，组织多个 case	`banetask project.projbt`
`.btlib`	可参数化任务或文本片段库	`&INCLUDE`
`.btder`	与同目录唯一 `.bt` 配对的独立 Derive DSL	`banetask file.btder`
`.btplt`	与同目录唯一 `.bt` 配对的独立 Plot DSL	`banetask file.btplt`
`.bwrk`	已准备 workflow	`btrun`
`.btdb`	单文件项目或 case 归档	`btdb archive ...`
`banetask.conf`	全局路径与默认值	启动时读取
`<program>.conf`	程序运行环境和命令模板	workflow 运行时读取
`bt.status`	case 中的任务状态	`banetask`、`btrun` 更新

6.2 文件级声明索引

声明	作用域	生效阶段	重复规则
普通 YAML metadata	文件	文件解析	同名键按 YAML 合并规则处理
`define:`	文件	任务展开	同名变量由更具体来源覆盖
`matrix:`	文件	任务展开	各轴生成笛卡尔积
结构化 YAML 集合	文件	foreach 展开	集合元素按声明顺序处理
`&INCLUDE`	文件或片段位置	文件解析	按出现顺序并入
`@foreach` / `@foreach?`	文本块	DSL 解析前	每个集合元素复制循环体
`@derive ... @end`	项目	结果处理后	名称必须唯一
`@plot ... @end`	项目	结果处理后	名称必须唯一
`autorun`	文件	workflow 收尾	后写值覆盖前值
`btrun:`	文件	workflow 生成	扁平化为路由提示

声明	作用域	生效阶段	重复规则
`totcore` / `totmem`	文件	任务准备	任务 `%control` 覆盖文件默认值
`finalize`	文件	项目收尾	按声明顺序执行

6.3 任务指令索引

指令	作用	主要阶段	可重复
`%source`	选择几何、上游任务、帧或结构化资源	任务准备	视 source 类型而定
`%program`	选择程序后端	任务准备	否
`%interface`	声明跨程序 provide / consume 接口	输入生成	按接口角色
`%batch`	把多帧输入展开为批处理	任务准备	否
`%control`	资源、guess、几何与运行控制	任务准备	子键后写覆盖
`%when`	声明准备条件	任务准备	条件按块组合
`%keywords`	程序关键词或规划 token	输入生成	后端决定合并规则
`%extrainput`	程序原生附加输入	输入生成	文本按声明顺序合并
`%template`	选择 `other` 模板	输入生成	否
`%param`	向模板或后端传参	输入生成	同名参数后写覆盖
`%args`	程序命令行参数	workflow 生成	按文本合并
`%runner`	选择 `script` 解释器	输入生成	否
`%body`	`script` 正文	输入生成	否
`%preprocess`	主程序前的 Process DSL 或命令	workflow 生成	按顺序追加
`%process`	主程序后的 Process DSL 或命令	workflow 生成	按顺序追加
`%mod`	修改几何、电荷、自旋、冻结与分层信息	任务准备	按声明顺序应用
`%meta`	任务 metadata 与标识	任务准备	同名键后写覆盖
`%plot`	任务内 Plot 声明	Plot	可有多个命名图

`%extrainputs` 是 `%extrainput` 的输入别名，不是独立指令。

6.4 变量、占位符与展开阶段

形式	来源	展开阶段	缺失行为
`[name]`	`define`、matrix、foreach、任务上下文	文本/任务展开	未定义时报告变量错误或按所在语法保留供后续阶段
`[item.field]`	foreach 集合元素	foreach 预展开	字段缺失时报错
`[item.index]`	foreach 自动字段	foreach 预展开	从 1 开始
`{{ field }}`	结果或状态快照	条件、Derive、Plot	按调用点的 missing 策略处理
`{{ derived(...) }}`	Derive 结果	条件或后续分析	依赖未生成时报缺失
`${NAME}`	配置与 workflow 环境	配置读取或 shell 执行	由配置解析器或 shell 规则决定
`[xyz_file]`	输入生成器导出的几何文件	输入生成	被引用时保留；后端不支持或 source 缺失时报错
`[geom]`	几何正文	输入生成	source 缺失时报错
`[charge]`	最终总电荷	输入生成	使用 source / mod 的最终值
`[spin]`	最终自旋多重度	输入生成	使用 source / mod 的最终值
`[uhf]`	未成对电子数	输入生成	计算为 `spin - 1`
`${ACTUAL_INPUT_FILE}`	workflow 实际输入	运行	manifest 必须提供
`${ACTUAL_OUTPUT_FILE}`	workflow 实际输出	运行	manifest 或配置必须提供
`${ARGS}`	`%args`	workflow 命令生成	未设置时为空

6.5 配置查找顺序速查

对象	从高到低的优先级
全局配置值	进程环境变量；显式配置文件；用户配置；安装配置；内置默认值
程序环境配置目录	CLI `--config-dir`；`BTRUN_TASK_CONFIG_DIR`；`BANETASK_ENVCONF_PATH`；安装目录
execution profile	显式 profile；target/route 指定 profile；用户 profile；安装 profile
`other` 模板目录	任务 `TEMPLATE_DIR`；调用方显式目录；`BANETASK_OTHER_TEMPLATES_PATH`
task 文件	CLI 路径；当前目录；`BANETASK_TASK_PATH`

对象	从高到低的优先级
source 文件	任务目录、case 目录与配置的 source 搜索路径，按 `%source` 类型解析

6.6 生成文件生命周期

文件或目录	生成者	消费者	可否重建
程序输入	后端输入生成器	主程序	可由任务文件与 source 重建
主 source stage XYZ	source 解析器	XYZ 输入、模板、命令或 source 恢复	未被引用且非恢复快照时自动清理；其余可由 source 重建或必须保留
`task.manifest.json`	`banetask`	workflow、结果处理、归档	可重建
`pre_comd`	`banetask`	workflow	可重建
`comd`	`banetask`	workflow	可重建
`.bwrk`	`banetask`	`btrun`	可重建
`bt.status`	`banetask` / `btrun`	调度与诊断	不应手工伪造
主输出与程序 sidecar	外部程序	结果解析器、用户	原始计算产物
`.bane.taskmeta.json`	结果处理	record、数据库	可由 manifest 与任务上下文重建
`.bane.result.user.kv`	用户 Process DSL	快照合并器	仅在用户命令可重复时可重建
`.bane.result.kv`	结果处理	Derive、Plot、查询	可由受支持的原始输出重建
`records/*.json`	发布流程	数据库和归档	可由 revision 与对象重建
`revisions/<run_id>/*.json`	发布流程	provenance、历史查询	应保留
`.bane/store/objects/sha256/`	artifact 存储	record、归档	只有源文件仍在时可重建
`project.db`	数据库同步	`btdb`、Derive、Plot	可由完整 records/revisions 重建
`Results/`	Process、Derive、Plot	用户和归档	取决于声明与原始结果

文件或目录	生成者	消费者	可否重建
`.btdb`	`btdb archive`	校验、恢复、导入	是结项归档，不从缺失原始对象推断

6.7 状态索引

状态或阶段	含义	下游可否继续
prepared / rendered	输入和 workflow 已生成	可提交
queued	已进入本地或调度器队列	等待
running	workflow 正在执行	否
success	preprocess、主命令、process 满足成功条件	是
success_with_warnings	主流程成功，允许忽略的 process 错误已记录	取决于 autorun 策略
continued_with_errors	`autorun: always` 保留错误并继续	是，但必须检查失败阶段
failed	某一必要阶段非零退出或结果条件不满足	否
cancelled	用户或调度器取消	否
skipped	条件不满足或依赖不可用	由跳过原因决定
source_pending	上游 source 尚未完成	等待上游
configuration_error	配置、profile、target 或程序环境缺失	否

具体状态字段、阶段退出码和已有状态文件的读取规则见第 4 章“`bt.status`”。

6.8 Process DSL 指令索引

指令	作用
`run`	执行外部命令
`mwfn` / `multiwfn`	调用 Multiwfn 模板
`publish`	发布文件或结果对象
`cp`	复制文件
`mv`	移动文件
`mkdir` / `mkdirp`	创建目录
`snapshot` / `mass`	创建结果快照或批量快照
`keyword`	修改受支持的输入关键词
`result`	抽取并发布程序结果

指令	作用
`dbcollect`	收集结果到数据库队列
`atomvec`	生成或导入逐原子向量
`banewfn`	运行 BaneWfn 分析
`benv`	在 Bane 环境中执行命令
`fork`	从结构文件生成可供 foreach 使用的集合
原样命令	交给所选 shell 执行

6.8.1 Process DSL 常用选项

指令	选项	作用
`result` / `dbcollect`	`-k`, `--key <name>`	指定写入 KV 的键
`result` / `dbcollect`	`--merge-kv <file>`	合并一个 KV 文件
`result` / `dbcollect`	`--exec-kv <command...>`	执行命令并按 KV 读取输出
`mkdir` / `mkdirp`	`-p`, `--parents`	接受父目录创建形式；实现始终按递归创建目录
`keyword`（ORCA 子项）	occurrence 路径	生成底层工具的 `--occ`，用户在 DSL 中通过路径 occurrence 表达

6.9 Derive 语句与函数索引

类别	条目
标量声明	`let`、`require`、`assert`、标量 `emit`、`report`
表来源	`from tasks(...)`、`from derived(...)`、`from atoms(...)`
表变换	`join`、`where`、`select`、`group by`、`aggregate`、`reduce`、`sort`、`limit`
sink	`emit table`、`emit atomvec`、`emit kv`、`emit artifact`
结果函数	规范属性函数、`energy()`、`single_point()`、`result()`、`kv()`、`derived()`
数值函数	`convert()`、`min()`、`max()`、`sort()`、`filter()`、`at()`、`nth()`、`abs()`
存在性与索引	`present()`、`exists()`、`coalesce()`、`row_index()`、`task_index()`、`matrix_value()`
热化学	RRHO/qRRHO、组合自由能及单位安全运算函数

6.10 Plot 语句与图类型索引

类别	条目
声明位置	任务内 `%plot`；项目级 `@plot ... @end`；独立 `.btplt`
数据	任务结果、Derive table/value、显式轻量快照
通用语句	data/source、x/y、filter、group、label、unit、style、output、missing policy
图类型	`barrier`、`convergence`、`curve`、`spectrum`、`scatter`、`correlation`、`heatmap`
统计图	`histogram` / `hist`、`bar` / `barchart`、`errorbar` / `errorbars`
量化图	`frontier` / `fmo`、`excited` / `excited-state`

6.11 程序名与输入别名

输入	规范名称或写法
`g16`	任务文件使用 `%program gaussian`
`%extrainputs`	`%extrainput`
`job=<name>`（`other` 参数）	`%template <name>`
`%keywords` 中的 script runner	`%runner`
未加花括号的 `$NAME` 配置变量	`${NAME}`
`backend` / `kind` profile 字段	`scheduler`

6.12 CLI 入口索引

命令	主要用途
`banetask`	准备任务，执行 Derive/Plot
`btrun init`	初始化运行配置
`btrun setup`	创建 target、route、queue 和 profile 配置
`btrun kick`	扫描并提交可运行任务
`btrun task`	运行单个 `.bwrk` 或任务目录
`btrun list`	查看本地与远程队列
`btrun cancel`	取消任务
`btdb case` / `project`	查询和同步数据库
`btdb archive`	创建、校验、恢复 `.btdb`
`btproc result`	抽取程序结果
`btproc fork collect`	收集多帧结构与轨迹元数据
`btproc atomvec`	计算并导出逐原子向量
`btproc method`	生成方法学说明并检查参考库

命令	主要用途
`btproc fcclasses report`	生成 FCclasses 任务、case 或 project 报告
`btool`	模板、格式、迁移、展开、解释和 lint
`btc`	交互式输入生成
`var`	变量读取、修改与 CSV 导出

6.13 错误类别与退出行为

类别	发生阶段	处理
语法错误	文件解析	不生成任务；报告文件、行号和附近语句
展开错误	include/matrix/foreach	不生成受影响任务；报告缺失变量、集合或循环边界
source 错误	任务准备	任务等待、跳过或失败，取决于 source 是否尚可能完成
输入生成错误	后端渲染	不写出可提交 workflow
配置错误	workflow 生成或提交	不提交；报告缺失键、profile、target 或命令模板
调度错误	`btrun`	保留提交命令、队列输出和 backend 返回码
程序错误	主命令	记录主退出码和输出；不把后处理成功视为主程序成功
结果错误	process / publish	按 autorun 策略标记失败、警告或继续
Derive 错误	Derive	报告缺失属性、单位不兼容、类型或表达式错误
Plot 错误	Plot	报告数据、列、单位、图类型或渲染器错误
归档错误	`.btdb`	校验失败时不把不完整容器视为有效归档

6.14 术语表

术语	定义
canonical property	属性注册表中的唯一规范名称
alias	解析到 canonical name 的输入名称
profile	调度器或运行后端的一组提交参数
target	一个可执行任务的本地、调度器或远程目标
route	根据任务资源和标签选择 target/profile 的规则
staging	把输入、依赖和脚本组织到任务目录
provenance	结果、输入、命令、环境与 artifact 的来源关系
record	一次发布后的任务结果记录

术语	定义
revision	同一任务不同运行或重新解析的版本记录
object	以内容哈希寻址的 artifact blob
requirement	查询某属性所需的输出分析能力

6.15 配置键索引

6.15.1 BaneTask 全局配置

下列键可写入 `banetask.conf`；同名进程环境变量覆盖文件值。完整查找顺序、路径分隔符和默认行为见第 4 章。

配置键	用途分组
`BANETASK_TASK_PATH`	路径与工具
`BANETASK_LOG_PATH`	路径与工具
`BANETASK_SCRIPTS_PATH`	路径与工具
`BANETASK_WFN_EXAMPLES_PATH`	路径与工具
`BANETASK_TEMPLATE_PATH`	路径与工具
`BANETASK_BTLIB_PATH`	路径与工具
`BANETASK_OTHER_TEMPLATES_PATH`	路径与工具
`BANETASK_EXTERNAL_SCRIPTS_PATH`	路径与工具
`BANETASK_BANEWFN_PATH`	路径与工具
`BANETASK_BTDB_PATH`	路径与工具
`BTMETHOD_REFS_PATH`	数据库与方法
`BANETASK_GNUPLOT`	Plot
`BANETASK_ENVCONF_PATH`	路径与工具
`BTRUN_DEFAULT_BACKEND`	btrun
`BTRUN_SELECTION_MODE`	btrun
`BTRUN_ALLOW_AUTO_REMOTE`	btrun

6.15.2 程序环境配置

配置键	出现于示例配置	作用
`BDF_TMPDIR`	`bdf.conf`	程序或站点配置参数

配置键	出现于示例配置	作用
CONF_CORES	banewfn.conf, bdf.conf, comd.conf, fc.conf, fcclasses.conf, g16.conf, gamess.conf, gxss.conf, maple.conf, mokit.conf, mrcc.conf, multiwfn.conf, nwchem.conf, orca_610.conf, xtb.conf	默认总核心数
CONF_MEMORY	banewfn.conf, comd.conf, fc.conf, fcclasses.conf, g16.conf, gamess.conf, gxss.conf, maple.conf, mokit.conf, mrcc.conf, multiwfn.conf, nwchem.conf, orca_610.conf, xtb.conf	默认总内存
CONF_SPAN_NODES	g16.conf	程序或站点配置参数
DESCRIPTION	comd.conf, fc.conf	程序或站点配置参数
ENV_SETUP	comd.conf, fcclasses.conf	程序或站点配置参数
GAUSS_SCRDIR	mokit.conf	程序或站点配置参数
GMS	mokit.conf	程序或站点配置参数
LD_LIBRARY_PATH	nwchem.conf	程序或站点配置参数
MKL_NUM_THREADS	mrcc.conf, xtb.conf	程序或站点配置参数
MULTIWFN_PATH	banewfn.conf, multiwfn.conf	程序或站点配置参数
Multiwfnpath	banewfn.conf, multiwfn.conf	程序或站点配置参数
OMPI_MCA_btl_openib_allow_ib	orca.conf, orca_610.conf	程序或站点配置参数
OMPI_MCA_btl_openib_warn_default_gid_prefix	orca.conf, orca_610.conf	程序或站点配置参数
OMP_NUM_THREADS	bdf.conf, mrcc.conf, xtb.conf	程序或站点配置参数
OMP_PLACES	mrcc.conf	程序或站点配置参数
OMP_PROC_BIND	mrcc.conf	程序或站点配置参数
OMP_STACKSIZE	banewfn.conf, bdf.conf, multiwfn.conf, xtb.conf	程序或站点配置参数
OUTPUT_SUFFIX	amesp.conf, bdf.conf, comd.conf, fcclasses.conf, gamess.conf, maple.conf, mopac.conf, mrcc.conf	主输出后缀
PATH	PATH.conf, banewfn.conf, fc.conf, g16.conf, gamess.conf, multiwfn.conf, nwchem.conf	程序或站点配置参数

配置键	出现于示例配置	作用
PGI_FASTMATH_CPU	g16.conf, mokit.conf	程序或站点配置参数
RAW_MODE	fc.conf	程序或站点配置参数
RUN_CMD_TEMPLATE	comd.conf, fcclasses.conf	主命令模板
INPUT_FILENAME	comd.conf, mrcc.conf	精确匹配的主输入 basename；与 SUFFIX 互斥
SUFFIX	amesp.conf, banewfn.conf, bdf.conf, fc.conf, fcclasses.conf, g16.conf, gamess.conf, gxss.conf, maple.conf, mokit.conf, mopac.conf, multiwfn.conf, nwchem.conf, orca.conf, orca_610.conf, xtb.conf	主输入后缀；与 INPUT_FILENAME 互斥
UCX_IB_ADDR_TYPE	orca.conf, orca_610.conf	程序或站点配置参数
USE_ABSOLUTE_PATH	mokit.conf, nwchem.conf	程序或站点配置参数
g16root	mokit.conf	程序或站点配置参数

6.16 规范结果属性注册表

结果查询、Derive 和 Plot 共用下列规范属性。canonical name 是查询键；alias 解析到同一属性。缺失属性返回 unavailable 或 missing，不以零值替代。

6.16.1 qc.atom.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.atom.gradient.x	atom.gradient.x	序列	原子	力 / hartree/bohr	gradient	standard	Final Cartesian gradient x component aligned by atom.
qc.atom.gradient.y	atom.gradient.y	序列	原子	力 / hartree/bohr	gradient	standard	Final Cartesian gradient y component aligned by atom.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.atom.gradient.z	atom.gradient.z	序列	原子	力 / hartree/bohr	gradient	standard	Final Cartesian gradient z component aligned by atom.
qc.atom.charge.mulliken	atom.charge.mulliken, mulliken_charge	序列	原子	无量纲	charge	standard	Mulliken atomic charge aligned by atom id.
qc.atom.charge.loewdin	atom.charge.loewdin, atom.charge.lowdin, loewdin_charge, lowdin_charge	序列	原子	无量纲	charge	standard	Loewdin/Lowdin atomic charge aligned by atom id.
qc.atom.population.mulliken	atom.population.mulliken, mulliken_population	序列	原子	无量纲	charge	standard	Mulliken atomic population aligned by atom id.
qc.atom.population.loewdin	atom.population.loewdin, atom.population.lowdin, loewdin_population, lowdin_population	序列	原子	无量纲	charge	standard	Loewdin/Lowdin atomic population aligned by atom id.
qc.atom.spin_population.mulliken	atom.spin_population.mulliken, mulliken_spin_population	序列	原子	无量纲	charge	standard	Mulliken atomic spin population aligned by atom id.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.atom.spin_population.loewdin	atom.spin_population.loewdin, atom.spin_population.lowdin, loewdin_spin_population, lowdin_spin_population	序列	原子	无量纲	charge	standard	Loewdin/Lowdin atomic spin population aligned by atom id.

6.16.2 qc.basis.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.basis	basis, basis_set, qc.basis_set	文本标量	作业	未指定	-	summary	Basis set assigned to the final calculation job.

6.16.3 qc.charge.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.charge	charge, molecular_charge	标量	无	无量纲	-	summary	Total molecular charge of the final chemical system.

6.16.4 qc.convergence.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.convergence.status	convergence.status, opt.converged	文本标量	步骤	未指定	gradient	standard	Convergence status of the final calculation step.

6.16.5 qc.dipole.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.dipole.x	dipole.x	标量	无	DipoleMoment / a.u.	dipole	standard	Permanent dipole x component in atomic units.
qc.dipole.y	dipole.y	标量	无	DipoleMoment / a.u.	dipole	standard	Permanent dipole y component in atomic units.
qc.dipole.z	dipole.z	标量	无	DipoleMoment / a.u.	dipole	standard	Permanent dipole z component in atomic units.
qc.dipole.magnitude	dipole.magnitude	标量	无	DipoleMoment / a.u.	dipole	standard	Permanent dipole magnitude in atomic units.
qc.dipole.debye	dipole.debye, dipole, dipole.magnitude_debye	标量	无	DipoleMoment / debye	dipole	standard	Permanent dipole magnitude in debye.

6.16.6 qc.energy.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.energy.value	energy.value, energy_component, selected_energy	标量	无	能量 / hartree	-	standard	A canonical energy record selected by method, quantity, spin component, orbital reference, state, job, step, or coarse energy kind.
qc.energy.primary	energy, primary_energy	标量	无	能量 / hartree	-	summary	Final preferred energy-like result (for MOPAC this may be a standard heat of formation).
qc.energy.scf	scf_energy	标量	无	能量 / hartree	-	summary	Final SCF energy.
qc.energy.correlated	correlated_energy, post_scf_energy, post_hf_energy, posthf_energy	标量	无	能量 / hartree	-	summary	Final correlated/post-SCF energy.
qc.energy.double_hybrid	double_hybrid_energy, doublehybrid_energy, dh_energy, double_hybrid, doublehybrid	标量	无	能量 / hartree	-	summary	Final double-hybrid DFT total energy, including the perturbative E2 correction.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.energy.composite	composite_energy, composite, cbs_qb3_energy, cbsqb3_energy	标量	无	能量 / hartree	-	summary	Final composite-method energy, such as CBS-QB3/G3/G4.
qc.energy.composite_zero_k	composite_zero_k_energy, composite_0k_energy, cbs_qb3_0k_energy, cbsqb3_0k_energy	标量	无	能量 / hartree	-	summary	Final 0 K composite-method energy.
qc.energy.formation_enthalpy	heat_of_formation, formation_enthalpy, delta_h_f, dhf	标量	无	能量 / hartree	-	summary	Standard heat/enthalpy of formation reported by a parameterized model such as MOPAC.

6.16.7 qc.excited_state.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.excited_state.index	excited_state.index, state.index	序列	ExcitedState	步骤编号	excited states	standard	Excited-state index in the final state block.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.excited_state.energy	qc.excited_state.excitation_energy, excited_state.energy, excitation_energy	序列	ExcitedState	能量 / hartree	excited states	standard	Excitation energy for each excited state.
qc.excited_state.wavelength	excited_state.wavelength, excitation_wavelength, wavelength	序列	ExcitedState	长度 / nm	excited states	standard	Excitation wavelength for each excited state.
qc.excited_state.oscillator_strength	excited_state.oscillator_strength, oscillator_strength	序列	ExcitedState	无量纲	excited states	standard	Oscillator strength for each excited state.
qc.excited_state.total_energy	excited_state.total_energy, state_total_energy	序列	ExcitedState	能量 / hartree	excited states	standard	Total energy reported for each excited state.
qc.excited_state.multiplicity	excited_state.multiplicity	序列	ExcitedState	无量纲	excited states	standard	Spin multiplicity for each excited state when reported.
qc.excited_state.spin_squared	excited_state.spin_squared, s2	序列	ExcitedState	无量纲	excited states	standard	<S^2> value for each excited state when reported.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.excited_state.singles_character	excited_state.singles_character	序列	ExcitedState	无量纲 / %	excited states	standard	Singles-character percentage for each excited state when reported.
qc.excited_state.major_contribution	excited_state.major_contribution	TextSeries	ExcitedState	文本指定	excited states	standard	Largest reported configuration contribution for each excited state.

6.16.8 qc.geometry.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.geometry.final	geometry.final, final_geometry, xyz, atomcoords	三维向量序列	原子	长度 / angstrom	-	summary	Final Cartesian geometry as atom symbols with Angstrom coordinates.

6.16.9 qc.gradient.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.gradient.norm	gradient.norm, final_gradient_norm	标量	步骤	力 / hartree/bohr	gradient	standard	Norm of the final Cartesian gradient.
qc.gradient.rms	gradient.rms, final_gradient_rms	标量	步骤	力 / hartree/bohr	gradient	standard	RMS of the final Cartesian gradient components.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.gradient.max	gradient.max, final_gradient_max	标量	步骤	力 / hartree/bohr	gradient	standard	Maximum absolute final Cartesian gradient component.

6.16.10 qc.hessian.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.hessian.dimension	hessian.dimension, cartesian_hessian.dimension, qc.hessian.dim	标量	无	无量纲	Hessian	standard	Dimension of the final dense Cartesian Hessian matrix.
qc.hessian.cartesian	qc.hessian, hessian, hessian.cartesian, cartesian_hessian, cartesian_hessian.values	序列	Hessian element	Force constant / hartree/bohr^2	Hessian	standard	Final dense Cartesian Hessian flattened in row-major order.

6.16.11 qc.keywords.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.keywords	keywords	文本标量	作业	未指定	-	summary	Raw route/input keywords for the final calculation job.

6.16.12 qc.method.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.method	method	文本标量	作业	未指定	-	summary	Method or functional assigned to the final calculation job.

6.16.13 qc.multiplicity.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.multiplicity	multiplicity, spin_multiplicity	标量	无	无量纲	-	summary	Spin multiplicity of the final chemical system.

6.16.14 qc.natoms.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.natoms	natoms, atom_count	标量	原子	无量纲	-	summary	Number of atoms in the final chemical system.

6.16.15 qc.optimization.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.optimization.step	qc.opt.step, opt.step, step.ordinal, step	序列	步骤	步骤编号	-	standard	Native optimization cycle or normalized step ordinal.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.optimization.energy	qc.opt.energy, opt.energy, step.energy	序列	步骤	能量 / hartree	-	standard	Preferred energy at each calculation step.
qc.optimization.max_force	qc.opt.max_force, opt.max_force, step.max_force, max_force	序列	步骤	力 / hartree/bohr	gradient	standard	Maximum force convergence criterion by step.
qc.optimization.rms_force	qc.opt.rms_force, opt.rms_force, step.rms_force, rms_force	序列	步骤	力 / hartree/bohr	gradient	standard	RMS force convergence criterion by step.
qc.optimization.max_displacement	qc.opt.max_displacement, opt.max_displacement, step.max_displacement, max_displacement	序列	步骤	长度 / bohr	gradient	standard	Maximum displacement convergence criterion by step.
qc.optimization.rms_displacement	qc.opt.rms_displacement, opt.rms_displacement, step.rms_displacement, rms_displacement	序列	步骤	长度 / bohr	gradient	standard	RMS displacement convergence criterion by step.

6.16.16 qc.orbital.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.orbital.homo_energy	orbital.homo_energy, orbital.homo, homo, homo_energy	标量	轨道	能量 / hartree	orbital	standard	Highest occupied molecular orbital energy.
qc.orbital.lumo_energy	orbital.lumo_energy, orbital.lumo, lumo, lumo_energy	标量	轨道	能量 / hartree	orbital	standard	Lowest unoccupied molecular orbital energy.
qc.orbital.gap	orbital.gap, homo_lumo_gap, gap	标量	轨道	能量 / hartree	orbital	standard	HOMO-LUMO gap computed as LUMO minus HOMO.
qc.orbital.homo_index	orbital.homo_index, homo_index	标量	轨道	无量纲	orbital	standard	Index of the selected HOMO.
qc.orbital.lumo_index	orbital.lumo_index, lumo_index	标量	轨道	无量纲	orbital	standard	Index of the selected LUMO.
qc.orbital.energy	orbital.energy	序列	轨道	能量 / hartree	orbital	standard	Orbital energy series for the selected spin/kind.
qc.orbital.occupation	orbital.occupation	序列	轨道	无量纲	orbital	standard	Orbital occupation series for the selected spin/kind.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.orbital.alpha.energy	orbital.alpha.energy, alpha.orbital.energy, alpha_orbital_energy, alpha_orbital.energy, orbital.energy.alpha	序列	轨道	能量 / hartree	orbital	standard	Alpha-orbital energy series. Closed-shell orbital sets are accepted as alpha-compatible.
qc.orbital.alpha.occupation	orbital.alpha.occupation, alpha.orbital.occupation, alpha_orbital_occupation, alpha_orbital.occupation, orbital.occupation.alpha	序列	轨道	无量纲	orbital	standard	Alpha-orbital occupation series. Closed-shell orbital sets are accepted as alpha-compatible.
qc.orbital.beta.energy	orbital.beta.energy, beta.orbital.energy, beta_orbital_energy, beta_orbital.energy, orbital.energy.beta	序列	轨道	能量 / hartree	orbital	standard	Beta-orbital energy series.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.orbital.beta.occupation	orbital.beta.occupation, beta.orbital.occupation, beta_orbital_occupation, beta_orbital.occupation, orbital.occupation.beta	序列	轨道	无量纲	orbital	standard	Beta-orbital occupation series.
qc.orbital.spin	orbital.spin	TextSeries	轨道	未指定	orbital	standard	Orbital spin channel series for the selected orbital set.

6.16.17 qc.program.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.program	program, qc.program.name	文本标量	无	未指定	-	summary	Program name reported by the normalized QC document.
qc.program.kind	program.kind	文本标量	无	未指定	-	summary	Canonical program family, such as gaussian, orca, or gamess.
qc.program.version	program.version	文本标量	无	未指定	-	summary	Program version string when present in the source output.

6.16.18 qc.spin_orbit_coupling.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.spin_orbit_coupling.triplet_index	spin_orbit_coupling.triplet_index, soc.triplet_index, soc.triplet	序列	SpinOrbitCoupling步骤编号	无量纲	SOC	standard	Triplet-state index for each preferred spin-orbit-coupling matrix element.
qc.spin_orbit_coupling.singlet_index	spin_orbit_coupling.singlet_index, soc.singlet_index, soc.singlet	序列	SpinOrbitCoupling步骤编号	无量纲	SOC	standard	Singlet-state index for each preferred spin-orbit-coupling matrix element.
qc.spin_orbit_coupling.magnitude	qc.spin_orbit_coupling.matrix, spin_orbit_coupling.magnitude, spin_orbit_coupling.matrix, soc.magnitude, soc.matrix	序列	SpinOrbitCoupling步骤编号	能量 / cm-1	SOC	standard	Magnitude of each preferred spin-orbit-coupling matrix element; M_S components are preferred over Cartesian XYZ components for the same state pair.

6.16.19 qc.task_kind.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.task_kind	task_kind, task.kind, run_type	文本标量	作业	未指定	-	summary	Primary task kind inferred for the final calculation job.

6.16.20 qc.termination.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.termination	termination, status, qc.status	文本标量	无	未指定	-	summary	Program termination kind.

6.16.21 qc.thermochemistry.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.thermochemistry.origin	thermo.origin, thermochemistry.origin	文本标量	无	未指定	vibrations	standard	Origin of the selected thermochemistry block: program_output or local_calculation.
qc.thermochemistry.model	thermo.model, thermochemistry.model	文本标量	无	未指定	vibrations	standard	Thermodynamic model used by a locally calculated thermochemistry block.
qc.thermochemistry.frequency_scale.zero_point	thermo.scale.zpe, zpe_frequency_scale	标量	无	无量纲	vibrations	standard	Frequency scale factor used for zero-point energy.
qc.thermochemistry.frequency_scale.thermal_enthalpy	thermo.scale.enthalpy, enthalpy_frequency_scale	标量	无	无量纲	vibrations	standard	Frequency scale factor used for vibrational thermal energy and enthalpy.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.thermochemistry.frequency_scale.entropy	thermo.scale.entropy, entropy_frequency_scale	标量	无	无量纲	vibrations	standard	Frequency scale factor used for vibrational entropy.
qc.thermochemistry.frequency_scale.heat_capacity	thermo.scale.heat_capacity, heat_capacity_frequency_scale	标量	无	无量纲	vibrations	standard	Frequency scale factor used for vibrational heat capacity.
qc.thermochemistry.low_frequency_cutoff	thermo.low_frequency_cutoff	标量	无	频率 / cm^-1	vibrations	standard	Low-frequency cutoff recorded for a local thermochemistry calculation.
qc.thermochemistry.interpolation_frequency	thermo.interpolation_frequency	标量	无	频率 / cm^-1	vibrations	standard	qRRHO interpolation frequency recorded for a local thermochemistry calculation.
qc.thermochemistry.imaginary_frequency_threshold	thermo.imaginary_frequency_threshold	标量	无	频率 / cm^-1	vibrations	standard	Magnitude threshold for converting small imaginary modes to positive frequencies.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.thermochemistry.rotational_symmetry_number	thermo.rotational_symmetry_number, thermo.sigma	标量	无	无量纲	vibrations	standard	Rotational symmetry number used by a local thermochemistry calculation.
qc.thermochemistry.electronic_energy	thermo.electronic_energy, thermochemistry.electronic_energy	标量	无	能量 / hartree	vibrations	standard	Electronic energy associated with the final thermochemistry block.
qc.thermochemistry.total_mass	thermo.total_mass, thermochemistry.total_mass, molecular_mass	标量	无	质量 / amu	vibrations	standard	Total molecular mass reported in the final thermochemistry block.
qc.thermochemistry.gibbs_energy	free_energy, gibbs_energy, gibbs_free_energy, qc.energy.gibbs_free	标量	无	能量 / hartree	vibrations	standard	Final Gibbs free energy. Composite-method free energy is used when present.
qc.thermochemistry.gibbs_correction	gibbs_corr, gibbs_correction	标量	无	能量 / hartree	vibrations	standard	Final thermal correction to Gibbs free energy.
qc.thermochemistry.zero_point_energy	zpe, zero_point_energy	标量	无	能量 / hartree	vibrations	standard	Final zero-point energy correction.
qc.thermochemistry.thermal_energy	thermal_energy	标量	无	能量 / hartree	vibrations	standard	Final thermal energy.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.thermochemistry.thermal_correction	thermal_corr, thermal_correction	标量	无	能量 / hartree	vibrations	standard	Final thermal correction to electronic energy.
qc.thermochemistry.enthalpy	enthalpy, enthalpy_energy, qc.energy.enthalpy	标量	无	能量 / hartree	vibrations	standard	Final enthalpy energy. Composite-method enthalpy is used when present.
qc.thermochemistry.enthalpy_correction	enthalpy_corr, enthalpy_correction	标量	无	能量 / hartree	vibrations	standard	Final thermal correction to enthalpy.
qc.thermochemistry.entropy_correction	entropy_corr, entropy_correction	标量	无	能量 / hartree	vibrations	standard	Final entropy contribution represented as energy.
qc.thermochemistry.entropy	entropy, thermo.entropy	标量	无	熵 / J/mol/K	vibrations	standard	Final molar entropy.
qc.thermochemistry.cv	cv, thermo.cv, constant_volume_heat_capacity	标量	无	热容 / J/mol/K	vibrations	standard	Final constant-volume molar heat capacity.
qc.thermochemistry.cp	cp, thermo.cp, constant_pressure_heat_capacity	标量	无	热容 / J/mol/K	vibrations	standard	Final constant-pressure molar heat capacity.
qc.thermochemistry.temperature	temperature, thermo.temperature	标量	无	温度 / K	vibrations	standard	Temperature of the final thermochemistry block.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.thermochemistry.pressure	pressure, thermo.pressure	标量	无	压力 / atm	vibration	standard	Pressure of the final thermochemistry block.

6.16.22 qc.vibration.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.vibration.frequency	vibration.frequency, frequency	序列	Vibrational mode	频率 / cm-1	vibration	standard	Vibrational frequencies in the final analysis.
qc.vibration.reduced_mass	vibration.reduced_mass, reduced_mass	序列	Vibrational mode	质量 / amu	vibration	standard	Reduced mass for each vibrational mode.
qc.vibration.force_constant	vibration.force_constant, force_constant	序列	Vibrational mode	Force constant / mDyne/angstrom	vibration	standard	Force constant for each vibrational mode.
qc.vibration.ir_intensity	vibration.ir_intensity, ir_intensity	序列	Vibrational mode	Specific intensity / km/mol	vibration	standard	IR intensity for each vibrational mode.
qc.vibration.raman_activity	vibration.raman_activity, raman_activity	序列	Vibrational mode	Raman activity / angstrom^4/amu	vibration	standard	Raman activity for each vibrational mode.
qc.vibration.imaginary_count	nimag, imag_freq, imaginary_frequency_count	标量	无	无量纲	vibration	standard	Number of imaginary vibrational modes in the final analysis.

6.16.23 qc.vibronic.*

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.vibronic.kind	vibronic.kind, fcht.kind	文本标量	无	未指定	vibration	standard	Absorption or emission kind of the final vibronic spectrum.
qc.vibronic.framework	vibronic.framework, fcht.framework	文本标量	无	未指定	vibration	standard	Framework reported for the final vibronic simulation.
qc.vibronic.spectroscopy	vibronic.spectroscopy, fcht.spectroscopy	文本标量	无	未指定	vibration	standard	Spectroscopy type reported for the final vibronic simulation.
qc.vibronic.model	vibronic.model, fcht.model	文本标量	无	未指定	vibration	standard	Vibronic model, such as ADIABATIC HESSIAN.
qc.vibronic.transition_moment_approximation	vibronic.transition_moment_approximation, fcht.approximation	文本标量	无	未指定	vibration	standard	Electronic transition-moment approximation (FC, HT, or FCHT).
qc.vibronic.temperature	vibronic.temperature, fcht.temperature	标量	无	温度 / K	vibration	standard	Simulation temperature for the final vibronic spectrum.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.vibronic.zero_zero_energy	vibronic.zero_zero_energy, fcht.zero_zero_energy, zero_zero_transition	标量	无	SpectralCoordinate / cm-1	vibration	standard	0-0 transition energy of the final vibronic spectrum.
qc.vibronic.line_width	vibronic.line_width, fcht.hwhm	标量	无	频率 / cm-1	vibration	standard	Half-width at half-maximum used to broaden the final vibronic spectrum.
qc.vibronic.transition.energy	vibronic.transition.energy, fcht.transition.energy	序列	VibronicTransition	频率 / cm-1	vibration	standard	Relative transition energy with respect to the 0-0 transition.
qc.vibronic.transition.intensity	vibronic.transition.intensity, fcht.transition.intensity	序列	VibronicTransition	Spectral intensity	vibration	standard	Line intensity for each printed vibronic transition.
qc.vibronic.transition.dipole_strength	vibronic.transition.dipole_strength, fcht.transition.dipole_strength	序列	VibronicTransition	无指定	vibration	standard	Raw Gaussian DipStr value for each printed vibronic transition.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.vibronic.transition.assignment	vibronic.transition.assignment, fcht.transition.assignment	TextSeries	VibronicTransition	未指定	vibration	standard	Initial-to-final vibrational-state assignment for each printed transition.
qc.vibronic.spectrum.energy	vibronic.spectrum.energy, fcht.spectrum.energy	序列	VibronicSpectralPoint	SpectralCoordinate / cm-1	vibration	standard	Energy grid of the broadened final vibronic spectrum.
qc.vibronic.spectrum.intensity_0k	vibronic.spectrum.intensity_0k, fcht.spectrum.intensity_0k	序列	VibronicSpectralPoint	SpectralIntensity	vibration	standard	0 K intensity column of the broadened final vibronic spectrum.
qc.vibronic.spectrum.intensity	vibronic.spectrum.intensity, fcht.spectrum.intensity	序列	VibronicSpectralPoint	SpectralIntensity	vibration	standard	Simulation-temperature intensity column, or the last available intensity column.
qc.vibronic.duschinsky.row	vibronic.duschinsky.row, duschinsky.row, fcht.j_row	序列	DuschinskyElement	无量纲	vibration	standard	One-based initial-state normal-mode row index for each Duschinsky J-matrix element.

Canonical name	Alias	Shape	Axis	Dimension / unit	Required analysis	Minimum profile	定义
qc.vibronic.duschinsky.column	vibronic.duschinsky.column, duschinsky.column, fcht.j_column	序列	Duschinsky 量纲	无量纲	vibration	standard	One-based final-state normal-mode column index for each Duschinsky J-matrix element.
qc.vibronic.duschinsky.matrix	vibronic.duschinsky.matrix, duschinsky.matrix, fcht.j_matrix	序列	Duschinsky 量纲	无量纲	vibration	standard	Duschinsky J matrix flattened in row-major order.
qc.vibronic.shift_vector	vibronic.shift_vector, fcht.shift_vector, fcht.k_vector	序列	Vibrational mode	无量纲	vibration	standard	Gaussian Shift Vector (K) indexed by initial-state normal mode.
qc.vibronic.huang_rhys	vibronic.huang_rhys, huang_rhys, fcht.huang_rhys	序列	Vibrational mode	无量纲	vibration	standard	Huang-Rhys factor indexed by vibrational mode.

6.17 命令行工具

BaneTask 的命令行工具分为生成与重建、workflow 提交与队列、项目批处理、模板转换、集合采集、结果提取和静态检查等几类。不同入口共享同一套目录约定、结果链路和 DSL 解释规则。

工具速览如下：

- **banetask**：单 case 解析 **.bt** / **.projbt**，生成输入、**pre_comd** / **comd** 与 **.bwrk**；典型入口：**banetask case.bt**、**banetask case.bt --comd-only**。
- **btrun**：消费 **.bwrk**，提交 / 渲染 / kick workflow，也支持基于 **<type>.conf** 的直接任务提交；典型入口：**btrun kick --path ./demo_project**。
- **btdb**：同步 **records/*.json**、执行 SQL、渲染模板报告、导出 Obsidian 笔记树、做项

目 compare；典型入口：`btdb case ...`、`btdb project ...`、`btdb compare ...`。
- `btool`：模板管理、project 批量维护、改名、格式化 / 迁移、片段展开、解释 DSL；典型入口：`btool project ...`、`btool template ...`、`btool expand ...`、`btool explain ...`。
- `btc`：从 `.btc` 模板快速起单 case 或 project 骨架；典型入口：`btc template.btc --geom a.xyz`。
- `btproc fork collect`：把多帧结构 / 轨迹采集进 `extra.yaml` 结构化集合；典型入口：`btproc fork collect xyz cluster.xyz --as cluster`。
- `btool lint`：对 `.bt` 文件做静态检查；典型入口：`btool lint ./cases`。
- `btproc result`：从日志抽取优化 / 频率 / TDDFT 结果并写入结果链路；典型入口：`btproc result extract opt.log ...`。
- `btproc atomvec`：执行 per-atom 向量派生并写入 `.bane.values.kv`，可导出 extxyz；典型入口：`btproc atomvec eval --dsl "charge.avg = mean(a:charge, b:charge) --out avg.extxyz"`。
- `btproc method`：从任务输入、manifest 和参考库生成方法学 Markdown / JSON；典型入口：`btproc method generate --path ./case --task-file case.bt`。
- `var`：读写 YAML 变量与 `.bt` 头部变量视图；典型入口：`var read define:func --file case.bt`。

6.17.1 `banetask`

`banetask` 解析 `.bt` / `.projbt`，生成量化程序输入、`pre_comd` / `comd` 和 `.bwrk`，并执行任务文件中声明的 Derive DSL 与 Plot DSL。独立 `.btder` / `.btplt` 脚本也由同一入口运行。

6.17.1.1 用法与输入路径

```
banetask [options] [path]
```

`path` 可取：

- `.bt` 或 `.projbt` 文件；
- 包含任务文件的目录；
- 独立 `.btder` 或 `.btplt` 文件。

省略 `path` 时使用 `BANETASK_TASK_PATH`，该配置为空时使用当前目录。目录输入会递归查找 `.bt`。命令行只应给出一个路径。

独立 `.btder` / `.btplt` 必须与一个完整 `.bt` 位于同一目录；该目录当前层只能有一个 `.bt`，否则无法确定配对任务文件。`.btder` 默认进入 derive-only 模式，`.btplt` 默认进入 plots-only 模式；`--explain-derives` 和 `--explain-plots` 分别改为只解释依赖。

6.17.1.2 通用选项

选项	行为
`-h`, `--help`	显示帮助。
`-v`, `--version`	显示版本与引用信息。

选项	行为
`-l <path>`, `--log <path>`	把本次处理写入一个显式日志文件。未指定时按 `BANETASK_LOG_PATH` 与逐任务日志规则处理。
`-d`, `--debug`	启用 DEBUG 控制台与文件日志；默认日志名为 `banetask_debug.log`。
`--quiet`	控制台只显示错误和最终摘要；文件日志规则不变。
`--no-color`	关闭 ANSI 颜色；环境变量 `NO_COLOR` 也会关闭颜色。
`--comd-only`	只刷新 `pre_comd` / `comd`；不生成或改写 `.bwrk`，不修改 `bt.status`。

6.17.1.3 Plot 选项

选项	行为
`--no-plots`	不分析或渲染 `%plot` / `@plot`。
`--plots-only`	只从已有结果生成图，不准备计算任务。
`--plot <name-or-id>`	只处理一个图。任务内图 ID 使用 `task:plot`；顶层图可用声明名。
`--rerender-plots`	忽略已有 plot fingerprint，重新调用 renderer。
`--explain-plots`	只输出图的任务依赖、属性需求和解析计划；不读取结果、不渲染。
`--gnuplot <path>`	指定 gnuplot 可执行文件，覆盖 `BANETASK_GNUPLOT`。

6.17.1.4 Derive 选项

选项	行为
`--no-derives`	不分析或执行 `@derive`。
`--derives-only`	只从已有任务结果执行 `@derive`，不准备计算任务。
`--derive <name-or-id>`	只执行一个 derived result block。
`--rerender-derives`	忽略已有 derive fingerprint，重新计算并写出 sink。
`--explain-derives`	只输出依赖、结果属性需求、表来源和 sink 计划；不读取结果、不写产物。

6.17.1.5 模式组合规则

下列组合会在执行前报错：

- `--no-plots` 与 `--plots-only`、`--explain-plots` 或 `--plot`；
- `--no-derives` 与 `--derives-only`、`--explain-derives`、`--rerender-derives` 或 `--derive`；

- **--comd-only** 与任何 plot-only、plot-explain、derive-only 或 derive-explain 模式；
- **--plots-only** 与 **--explain-plots**；
- **--derives-only** 与 **--explain-derives**；
- plot-only / plot-explain 模式与 derive-only / derive-explain 模式。

独立脚本还受以下限制：

- **.btder** 不能与 **--plots-only**、**--explain-plots**、**--rerender-plots** 或 **--plot** 组合；
- **.btplt** 不能与 **--derives-only**、**--explain-derives**、**--rerender-derives** 或 **--derive** 组合。

--plot、**--derive** 只限制对应 artifact；未进入 only/explain 模式时，普通任务准备仍按其余选项执行。fingerprint 命中时，普通执行会跳过未变化的 Derive/Plot；使用对应 **--rerender-*** 可强制重算。

6.17.1.6 示例

```
banetask case.bt
banetask ./project
banetask case.bt --comd-only --no-plots --no-derives
banetask case.bt --derives-only
banetask case.bt --derive activation --rerender-derives
banetask case.bt --explain-derives
banetask case.bt --plots-only --plot energy:barrier
banetask case.bt --explain-plots
banetask analysis.btder
banetask figure.btplt --rerender-plots
```

6.17.2 btrun

btrun 提交 **.bwrk** workflow、渲染调度脚本、直接提交程序输入，并管理本地文件队列和 SSH/rsync remote profile。调度选择以 execution target 为单位：先发现本地 runner、本机 scheduler 和 remote profile，再为每个 target 匹配 queue，最后确定 backend。

6.17.2.1 命令总览

```
btrun kick [options] [path]
btrun submit [options] [file.bwrk ...]
btrun render [options] [file.bwrk ...]
btrun task [options] [input_file|input_dir ...]

btrun queue run [options] [--once|-1]
btrun queue status
btrun queue list|ls [STATE|--state STATE] [--json] [--no-remote]
                  [--cached|--no-refresh|--refresh] [-o|--format FIELDS]
btrun queue sync|retrieve|pull <jobid...|all|--all> [options]
btrun queue cancel <jobid|prefix|all> [--force|-f]
btrun queue kill <jobid|prefix|all>

btrun status <local|slurm|sge|pbs> <jobid> [options]
```

```
btrun cancel <local|slurm|sge|pbs> <jobid> [--force|-f] [options]
btrun backend list
btrun init [options]
btrun setup [section [name]] [options]
btrun help [command|conf]
btrun <command> --help
btrun --version
```

kick 先在目标路径运行 **banetask**（可找到该程序时），再递归提交生成的 **.bwrk**。**submit** 直接消费 **.bwrk**；省略文件时递归扫描 **--path** 或当前目录。**render** 执行同样的 target / queue 选择并把生成脚本写到标准输出，但不提交。**task** 不需要 **.bwrk**，而是用 **<type>.conf**、输入文件和可选 **task.manifest.json** 现场构造 job。

6.17.2.2 提交与 target 选择选项

这些选项用于 **kick**、**submit**、**render**、**task**；**status**、**cancel**、**queue sync** 也接受其中与 backend/profile 定位有关的选项。

选项	行为
-b <name>, **--backend <name>**	backend 过滤器：**auto**、**local**、**slurm**、**sge**、**pbs**。**auto** 发现全部可用 target。
-L, **--local**	只保留本地文件队列 target。
-S, **--slurm**	只保留 Slurm target，包括 remote profile；本机无需安装 Slurm 客户端。
-E, **--sge**	只保留 SGE target。
-B, **--pbs**	只保留 PBS target。
-P <name>, **--profile <name>**	只使用一个 profile。支持 **profiles/<name>.yaml** 与 backend 子目录布局。
--profiles <a,b,...>	把多个 profile 作为候选池，并把选中的 profile 持久化为 case route。不能与 **--profile** 同时使用。
-D <dir>, **--profile-dir <dir>**	指定 profile 目录。
-q <name>, **--queue <name>**	要求 queue 名、alias 或 native partition 匹配。
--auto-target	允许在没有显式远程约束时自动选中 remote profile。
--no-auto-target	本次调用关闭由 **BTRUN_ALLOW_AUTO_REMOTE** 启用的远程自动选择。
--no-local	排除本地文件队列 target。
--selection-mode <mode>, **--selection <mode>**	候选排序：**local_first**、**remote_first**、**best_fit**、**first_runnable_by_priority**。
--health-check	选择前通过 SSH 探测 remote scheduler 的 submit 命令。
--no-health-check	本次调用关闭 profile 中的 remote health check。
--explain	输出 target 发现结果、queue 匹配、资源拒绝原因、即时可运行性和最终选择。

选项	行为
`-d`, `--dry-run`	完成解析、选择和脚本准备，但不提交。
`-k`, `--keep-script`	保留生成的 wrapper 或 scheduler submit script。
`-p <dir>`, `--path <dir>`	`kick` 的目标路径，或 `submit` / `render` / `task` 的扫描路径。

同一选项既接受空格分隔值，也接受 `--name=value`；带短名的选项还接受相应的 `-x=value`。

6.17.2.3 本地队列与日志选项

选项	行为
`-c <n>`, `--cores <n>`	`queue run` 的本地总核心数；在 `task` 中，未给 `--task-cores` 时作为每个 job 的核心数覆盖。
`--gpus <n>`	本地 runner 可调度 GPU 总数；在 `task` 中覆盖 job GPU 请求。
`--gpu-ids <ids>`	可见 GPU ID，例如 `0`、`0,1`。
`-w <sec>`, `--poll <sec>`	runner 轮询间隔，必须为正数，默认 5 秒。
`-F`, `--follow`, `--tail`	在当前终端跟随 local job 的 `.out` / `.err`。
`--verbose`	输出完整 job ID、工作目录、stdout/stderr 路径和完成来源。
`--detach`, `--daemon`, `--background`	`queue run` 以后台方式运行；用于 `kick` / `submit` / `task` 时，在可排队提交后启动分离 runner。
`--foreground`, `--no-detach`	关闭由别名或外层调用设置的 detach。

本地提交会在 job 工作目录留下 `.btq` workflow，便于提交后检查完整环境、资源头和执行命令。队列目录优先取 `BTRUN_QUEUE_DIR`，其次取 `BANETASKQ_DIR`，最后使用 `~/.bane/taskq`；其中按 `pending`、`running`、`done`、`failed`、`cancelled` 保存独立的运行副本和 job 记录。队列副本与工作目录中的 `.btq` 相互独立。

runner 默认使用紧凑事件行。`START` 行显示任务名、job ID 末段、已分配的 CPU/GPU 和 PID；`DONE` / `FAIL` 行显示退出码和耗时，失败任务还显示 stderr 文件名。例如：

```
[btrun] local queue: profile=default  pid=17916  capacity=8C/1G[0]  poll=5s
[btrun] START opt_ts_ts_guess [65aa78]  4C  pid=34864
[btrun] DONE  opt_ts_ts_guess [65aa78]  rc=0  31s
```

需要查看完整路径和进程回收来源时使用 `--verbose`。

6.17.2.4 `btrun task`

```
btrun task [submission-options] [task-options] [input_file|input_dir ...]
btrun task [submission-options] [task-options] --path <dir>
```

```
btrun task --list-types [--config-dir <dir>]
btrun task --help-type <type> [--config-dir <dir>]
```

6.17.2.4.1 task 选项

选项	行为
`--config-dir <dir>`	指定 `<type>.conf` 目录。
`-t <type>`, `--type <type>`	显式任务类型。
`-o`, `--orca`	`--type orca` 的别名。
`-g`, `--gaussian-origin`	`--type gaussian_origin` 的别名。
`-n <n>`, `--task-cores <n>`	每个 job 的 CPU 核心数。
`-m <mb>`, `--memory <mb>`, `--memory-mb <mb>`	每个 job 的总内存，单位 MB。
`-x <arg>`, `--extra-args <arg>`, `--extra <arg>`	写入 `${EXTRA_ARGS}` / `${ARGS}` 的附加参数字符串。
`--post-var <name>`, `--post <name>`	主命令后追加 `<type>.conf` 中同名变量定义的命令片段。
`-r`, `--raw-mode`, `--raw`	把每个输入目录作为一个 raw task，不按后缀扫描单个输入文件。
`--file-mode`, `--no-raw-mode`, `--no-raw`	强制普通输入文件模式，即使配置写了 `RAW_MODE=true`。
`-f`, `--force`	预期输出已存在时仍提交。
`-a`, `--recursive`	递归扫描输入目录；默认只扫描一层。
`-l`, `--list-types`	列出配置目录中的任务类型后退出。
`-H <type>`, `--help-type <type>`	输出该类型的说明和解析结果后退出。

任务类型按以下优先级确定：

1. `-t` / `--type`；
2. `task.manifest.json` 的 `program_params.prog`；
3. manifest 的 `program_type`；
4. `gaussian`。

`xtbgau` 作为 `xtb` 的输入别名。配置目录按以下顺序查找：

1. `--config-dir`；
2. `BTRUN_TASK_CONFIG_DIR` 路径列表；
3. `BANETASK_ENVCONF_PATH` 路径列表；
4. `./envs`、`./conf/envs`、安装目录候选和 `~/.bane/task/envs`。

manifest 可提供以下用户字段：

- 输入与输出：`input_files`、`output_file_name`；
- 程序：`program_type`、`program_params.prog`；
- 参数：`program_params.args`、`program_params.extra_args`；

- CPU：**nprocs**、**nproc**、**cores**、**cores_pre_node**、**cores_per_node**；
- 内存：**memory**、**mem**、**maxcore**；
- GPU：**gpus**、**gpu_mem**、**gpu_type**。

CLI 值覆盖 manifest 和配置默认值。没有 manifest 时，Gaussian / g16 输入中的 **%nprocshared**、**%nproc**、**%mem**，以及 ORCA 输入中的 **%pal**、**!PALN**、**%maxcore** 会用于资源推断。预期输出已经存在时不提交，并打印跳过摘要；**--force** 可覆盖。

6.17.2.4.2 <type>.conf 文件格式

<type>.conf 同时支持标量 **KEY=value** 和 raw section：

```
[main]
DESCRIPTION=Gaussian via g16
SUFFIX=gjf
OUTPUT_SUFFIX=log
CONF_CORES=32

[ENV_SETUP]
source "$HOME/apprepo/gaussian/16/env.sh"

[RUN_CMD_TEMPLATE]
g16 "${ACTUAL_INPUT_FILE}" > "${ACTUAL_OUTPUT_FILE}"
```

- **[main]** 下及 raw section 外的 **KEY=value** 解析为标量。行内 **#** 属于值的一部分。
- **[NAME]** 必须独占一行并从第 1 列开始；其后的原始多行文本成为变量 **NAME**，直到下一个 section。
- **ENV_SETUP** 在主命令前执行；无内容时可写 **:**。
- **RUN_CMD_TEMPLATE** 是主命令。
- 任意 **[NAME]** 都可由 **--post-var NAME** 追加。
- 单引号包围的多行标量也可用于 raw section 外的配置值。

常用键：

键	含义
DESCRIPTION	**--help-type** 显示的说明。
SUFFIX	文件模式按后缀扫描；与 **INPUT_FILENAME** 互斥，两者均缺失时默认 **inp**。
INPUT_FILENAME	文件模式按 basename 精确扫描固定文件名；与 **SUFFIX** 互斥。
OUTPUT_SUFFIX	预期输出后缀，默认 **log**。
ENV_SETUP	运行前环境初始化。
RUN_CMD_TEMPLATE	主命令模板。
CONF_CORES	默认每 job 核心数。
CONF_MEMORY	默认每 job 内存，单位 MB。
CONF_SPAN_NODES	程序是否允许跨节点；task 模式默认 false。

键	含义
`RAW_MODE`	是否默认使用目录级 raw task。
`ARRAY`	backend 支持时使用的 scheduler array 大小。
`USE_ABSOLUTE_PATH`	true 时模板接收解析后的完整输入/输出路径；false 时只传文件名。

模板变量使用 `${NAME}`；`$NAME` 也可识别。可用变量包括：

变量	值
`${ACTUAL_INPUT_FILE}`	按 `USE_ABSOLUTE_PATH` 处理后的输入路径或文件名。
`${ACTUAL_OUTPUT_FILE}`	按 `USE_ABSOLUTE_PATH` 处理后的输出路径或文件名。
`${ACTUAL_INPUT_FILENAME}`	输入文件名。
`${ACTUAL_OUTPUT_FILENAME}`	输出文件名。
`${ACTUAL_INPUT_STEM}`	去掉后缀的输入 stem。
`${FILE_NAME}`, `${BASENAME}`	输入 stem 的别名。
`${WORKDIR}`	job 工作目录。
`${CORES}`	有效每 job 核心数。
`${EXTRA_ARGS}`, `${ARGS}`	CLI 或 manifest 提供的附加参数。
`${TYPE}`	最终任务类型。

`btrun help conf` 会在终端输出同一格式参考。

6.17.2.5 本地与远程队列

6.17.2.5.1 `queue run` 与 `queue status`

```
btrun queue run [local-options] [profile-options] [--once|-1]
btrun queue status
```

`queue run` 启动一个 runner，按 CPU/GPU 容量启动可运行的 pending job、回收结束进程，并处理 remote record 的状态更新和拉取。队列为空后退出；`--once` / `-1` 表示完成一次处理循环后退出。`queue status` 只输出本地 `pending`、`running`、`done`、`failed`、`cancelled` 数量。

SSH/rsync remote record 以远端 `.bane/run/chain.status.json` 和其引用的 task status 为语义状态，不用 scheduler `query_cmd` 判定 workflow 是否完成。语义终态后，runner 可拉回 `Results/`、`.bane/store/`、`.bane/cache/snapshots/`、`.bane/run/` 及任务目录 `.env`。成功 workflow 需要 chain 完整结束；当前任务明确失败时可在 chain 文件尚未出现时进入失败终态。

6.17.2.5.2 queue list

```
btrun queue list|ls [STATE]
btrun queue list|ls [--state|-s STATE] [--json] [--no-remote]
                       [--cached|--no-refresh|--refresh]
                       [-o|--format FIELDS]
```

默认包含 local 与 remote record，并对 remote/running 做一次只更新元数据的状态刷新；不会在 list 阶段拉取 artifact。**--cached** / **--no-refresh** 只读取本地缓存，**--refresh** 显式开启刷新，**--no-remote** 排除 remote record。STATE 可写为第一个位置参数，也可用 **--state** / **-s**。

文本模式发现已完成且待拉取的 remote job 时，交互式终端会询问是否立即执行 sync；非交互式环境打印可执行的 **btrun queue sync ...**。**--json** 不触发该提示，且不能与 **--format** / **-o** 同时使用。

--format 接受预设或字段列表：

预设	字段
default	**id,state,backend,profile,pid,exit,name,workdir**
short	**id,state,pid,exit,name**
paths	**id,state,name,workdir,source**
wide	default 字段全部使用宽度 0，不截断。
all	**id,state,backend,profile,native,pid,exit,name,workdir,source, queue_file,remote,retrieve,enqueue,start,finish**

字段以逗号或空白分隔，可写 **field:WIDTH**；宽度 0 表示不截断，路径字段截断时保留尾部。字段名与别名：

规范字段	别名
id	**jobid**, **job_id**
state	**st**
backend	无
profile	无
native	**nativeid**, **native_job_id**
pid	无
exit	**exit_code**, **rc**
name	**job**, **job_name**
workdir	**work_dir**, **cwd**
source	**source_path**
queue_file	**queuefile**
remote	无

规范字段	别名
`retrieve`	`needs_retrieve`, `sync`
`enqueue`	`enqueue_time`
`start`	`start_time`
`finish`	`finish_time`

JSON 数组中的每个对象固定包含 `id`、`state`、`backend`、`profile`、`native_job_id`、`job_name`、`workdir`、`source_path`、`queue_file`、`remote`、`needs_retrieve`、`retrieve_pending`、`retrieved`；有值时还包含 `semantic_message`、`last_error`、`pid`、`exit_code`、`enqueue_time`、`start_time`、`finish_time`。

6.17.2.5.3 `queue sync`、取消与终止

```
btrun queue sync|retrieve|pull <jobid...|all|--all>
    [-b|--backend <backend>] [-P|--profile <profile>] [-D|--profile-dir <dir>]
btrun queue cancel <jobid|prefix|all> [--force|-f]
btrun queue kill <jobid|prefix|all>
```

`sync`、`retrieve`、`pull` 等价，用于拉取满足过滤条件且待同步的 remote job。给出一个或多个完整 ID / 前缀时只处理匹配记录；`all` / `--all` 处理全部待同步记录。`queue cancel` 取消匹配的 local 排队或运行记录；`--force` 允许强制处理。`queue kill` 等同于强制取消。

顶层 `btrun status <backend> <jobid>` 与 `btrun cancel <backend> <jobid>` 直接调用所选 backend/profile 的查询或取消命令；remote profile 的显式 `status` 可以使用 `query_cmd`，与 queue runner 的语义状态判断不同。

6.17.2.6 `btrun init`

```
btrun init [options]
```

该命令探测本机 scheduler queue/partition 与本地 CPU、内存、GPU，生成 scheduler、profile、queue、route 及 `profiles/local.yaml`。默认配置根为 `~/.bane/task/btrun`。

选项	行为
`-O <dir>`, `--output-dir <dir>`	配置根目录。
`-P <name>`, `--profile <name>`	profile / cluster 名；未给出时交互询问，非交互回退到 `default`。
`-b <name>`, `--backend <name>`, `--scheduler <name>`	`slurm`、`sge`、`pbs` 或 `local`；默认 `slurm`。
`-S`, `--slurm`	选择 Slurm。
`-E`, `--sge`	选择 SGE。
`-B`, `--pbs`	选择 PBS。
`-L`, `--local`	不带值时选择 local backend；`--local=<mode>` 则设置 local target mode。

选项	行为
`--queues <a,b,...>`	直接给出 queue / partition 名并跳过 queue 名发现。
`--no-discover`	不运行 scheduler discovery 命令。
`--local-target <mode>`, `--local-execution <mode>`, `--local <mode>`	`auto`、`enable`、`disable`。
`--enable-local`	在生成的 `profiles/local.yaml` 中启用 local target。
`--disable-local`, `--no-local`	禁用 local target。
`--local-cores <n>`	覆盖探测到的本地 CPU 核心数。
`--local-memory <size>`, `--local-memory-mb <size>`	覆盖本地内存；接受 MB 数值或 `256G`、`262144M` 等单位写法。
`--local-gpus <n>`	覆盖 GPU 数量。
`--local-gpu-ids <ids>`	覆盖 GPU ID 列表。
`--local-reserve-cores <n>`	从 local 调度容量中保留核心。
`--local-reserve-memory <size>`, `--local-reserve-memory-mb <size>`	从 local 调度容量中保留内存。
`--local-reserve-gpus <n>`	从 local 调度容量中保留 GPU。
`-y`, `--yes`	非交互接受探测值和默认值。
`-f`, `--force`	覆盖已有生成文件。
`-d`, `--dry-run`	打印将生成的文件，不写磁盘。

检测到本机 Slurm/SGE/PBS 客户端时，`auto` 通常把机器视为登录节点并禁用 local target；未检测到本机 scheduler 且资源可用时启用 local target。`BTRUN_INIT_SKIP_GPU_PROBE` 或 `BTRUN_SKIP_GPU_PROBE` 为真时不运行 `nvidia-smi`，仍可用 CLI 显式填写 GPU。

6.17.2.7 `btrun setup`

`btrun setup` 按分区查看或编辑运行配置。可用分区为 `global`、`task`、`scheduler`、`profile`、`queue`、`route`、`local`、`paths` 和 `init`；不带分区时打开交互式菜单。目标文件默认可从起始模板创建，`--no-create` 要求文件必须已经存在；`--print-path` 只显示路径，`--list` 列出可用分区或已有配置。路径覆盖、编辑器和非交互选项见第 4 章“`btrun setup` 分区配置”及 `btrun setup --help`。

```
btrun setup
btrun setup task orca
btrun setup queue cluster --editor "code --wait"
btrun setup paths
btrun setup init --yes -P cluster --backend slurm
```

6.17.2.8 target、route 与资源排序

- `-S` / `-E` / `-B` 是 backend 过滤器，不要求本机存在对应 scheduler；remote profile 仍可入选。

- 未显式限制 backend/profile 时，remote profile 默认只参与发现；`--auto-target` 或 `BTRUN_ALLOW_AUTO_REMOTE=true` 才允许自动中选。
- local target 只有在 `profiles/local.yaml` 或显式 profile 中 `local.enabled: true` 时参与；可用容量会扣除 reserve 和当前 `running/` 记录。
- 总容量满足但暂时繁忙的 target 可作为排队候选；可以立即运行的 target 优先。
- `--profiles` 候选在资源相同时使用 case 稳定哈希分流。首次选中后，case 根写入 `.btrun-route.json`，后续在同一候选池中复用。
- case 可直接提供 `.btrun-route.json`、`.btrun-route.yaml`、`.btrun-route.yml`，或在 `extra.yaml` / `extra.yml` 中写 `btrun:` / `btrun_route:`。

约束优先级为：CLI backend/profile/queue，其次 `.bwrk` 头部 backend/profile，再次 case route，最后 `--profiles` 候选池。硬过滤完成后，queue 资源、selection mode、profile/queue priority 与稳定 tie-breaker 决定 target。`BTRUN_DEFAULT_BACKEND` 只提供 backend 偏好。

```yaml
btrun:
  backend: slurm
  profile: beta
```

6.17.2.9 job 目录 `.env`

Bash local wrapper 与 scheduler script 在运行正文前读取任务目录 `.env`。用户区可写普通赋值、`export` 和 `unset`；btrun 自动区由 runner 原子更新，并用实际分配覆盖 `CORES`、`BTRUN_JOBID`、GPU 数量、GPU ID 和设备可见性。调度内部容量变量只存在于当前 job 进程，不写回 `.env`。Windows Cmd wrapper 不读取或更新该文件。

6.17.2.10 示例

```
btrun kick --path ./demo_project
btrun submit -S -P cluster ./case/opt/opt.bwrk
btrun render --auto-target --selection-mode best_fit --explain
 ↪  ./case/opt/opt.bwrk
btrun task -L -t gaussian --path ./cases -a -c 64 -n 16
btrun task -S -t orca mol1.inp mol2.inp
btrun task --path ./case-with-manifest
btrun task --list-types --config-dir ~/.bane/task/envs
btrun task --help-type gaussian
btrun queue run --cores 64 --gpus 2 --gpu-ids 0,1 --poll 5
btrun queue list --state done -o id,state,name,workdir:36
btrun queue list --json --cached
btrun queue sync all -P cluster
btrun queue cancel abc123 --force
btrun status slurm 123456 -P cluster
btrun cancel slurm 123456 -P cluster
btrun init --yes -P cluster --backend slurm --local-target disable
btrun help conf
```

6.17.3 `btdb`

`btdb` 管理 case / project 结果数据库、报告模板、工件恢复、重新解析、Obsidian 导出和不可变 `.btdb` 归档。数据库索引位于作用域根目录的 `.bane/cache/project.db`；持久事实仍

以 **.bane/store/records/**、**.bane/store/revisions/** 和 **.bane/store/objects/** 为准。

6.17.3.1 命令总览

```
btdb case <command> [options]
btdb project <command> [options]

btdb case|project init [--path PATH]
btdb case|project sync [--path PATH] [artifact-options]
btdb case|project rebuild [--path PATH] [artifact-options]
btdb case|project verify [--path PATH]
btdb case|project restore --out DIR [selectors] [--overwrite]
btdb case|project reparse [selectors] [reparse-options]
btdb case|project query [--sql SQL|SQL] [--format FORMAT]
btdb case|project export [--sql SQL|SQL] [--format FORMAT] [--out FILE]
btdb case|project archive create --out FILE.btdb [archive-options]

btdb case|project template list [--scope any|case|project]
btdb case|project template show NAME
btdb case|project template query NAME [--param KEY=VALUE]... [options]
btdb case|project template export NAME [--param KEY=VALUE]... [options]
btdb case|project obsidian [export] [options]

btdb archive info <FILE.btdb>|--path FILE.btdb|--archive FILE.btdb
btdb archive verify <FILE.btdb>|--path FILE.btdb|--archive FILE.btdb
btdb archive restore|import <FILE.btdb> --out DIR [options]

btdb compare <LEFT_TASK> <RIGHT_TASK> [options]
btdb project compare <LEFT_TASK> <RIGHT_TASK> [options]
```

templates 是 **template** 的别名；**btdb project compare** 是顶层 **btdb compare** 的别名。**obsidian export** 中的 **export** 可省略。

6.17.3.2 作用域与路径解析

- **case** 接受 case 根目录、case 内任意任务目录或 **.bt** 文件。工具会向上定位 case 根目录。
- **project** 接受 project 根目录、project 内任意路径或 **.projbt** 文件。工具会向上定位包含 **.projbt** 的 project 根目录。
- **--path** 可用于所有 case / project 命令。多数命令也接受一个不带选项名的位置路径；为避免 SQL、模板参数和路径混淆，自动化脚本宜显式使用 **--path**。
- **init** 只创建或迁移 SQLite schema。**sync** 执行增量同步；**rebuild** 删除派生 SQLite 索引后，从 current records、revisions 和可发现任务重新建立索引，不删除持久 records、revisions 或 objects。
- **query**、**export**、模板 query/export、Obsidian export 和 **compare** 会先同步当前作用域，使查询基于最新 current records 与已归档 revisions。

6.17.3.3 case / project 通用选项

选项	行为
`--path <path>`	指定 case / project 路径。
`--sql <sql>`	`query` / `export` 的 SQL。SQL 也可作为第一个位置参数给出。
`--format <format>`	`table`、`tsv`、`csv`、`json` 或 `markdown`。`query` 默认 `table`；`export` 默认 `csv`；模板命令默认使用模板自身声明的格式。
`--out <path>`	`export` / `template export` 的输出文件、`obsidian export` / `restore` 的目标目录，或 `archive create` 的 `.btdb` 输出文件。`export` 省略时按所选格式写到标准输出。
`--no-color`	接受该选项以便统一脚本调用；`btdb` 数据输出本身不依赖颜色。

`query` 和 `export` 必须提供非空 SQL。`export --format table` 可被解析，但文件导出宜使用 `tsv`、`csv`、`json` 或 `markdown`；需要终端表格时使用 `query`。

6.17.3.4 同步与工件归档选项

以下选项用于 `sync`、`rebuild`、查询前同步、模板/Obsidian 导出前同步和 `archive create` 的预同步：

选项	行为
`--artifacts <profile>`	工件档位：`none`、`minimal`、`standard`、`full`；默认 `standard`。
`--artifact-profile <profile>`	`--artifacts` 的别名。
`--include-native-wavefunctions`	在存在时额外归档 `.chk`、`.gbw`、`.rwf` 等原生波函数/检查点文件；这些文件不是成功归档的必需项。
`--max-artifact-bytes <N>`	跳过大于 `N` 字节的候选工件；`0` 表示不设大小上限。参数必须是非负整数。

6.17.3.5 已归档工件恢复

```
btdb case|project restore --out <dir>
    [--run-id <id>] [--case <key>|--case-key <key>]
    [--artifact <key-or-role>|--artifact-key <key-or-role>]
    [--overwrite] [--path <path>]
```

恢复结果按 case、task、run 和原始相对路径组织。选择器可组合：

- `--run-id` 只恢复一个 run；
- `--case` / `--case-key` 只恢复一个 case；
- `--artifact` / `--artifact-key` 按工件 key 或 role 过滤；该选项只对 `restore` 有效；
- 默认不覆盖已有文件；`--overwrite` 允许替换目标文件。

`restore` 会先同步数据库，然后从内容寻址对象存储恢复已经登记的工件。它不同于顶层 `btdb archive restore`：前者操作在线数据库，后者解包一个独立 `.btdb` 文件。

6.17.3.6 重新解析归档结果

```
btdb case|project reparse [--path <path>]
    [--run-id <id>] [--case <key>|--case-key <key>]
    [--result-profile summary|standard|full|--profile ...]
    [--no-opt] [--no-freq] [--no-tddft]
```

`reparse` 从已归档的 `output.primary` 运行当前结果解析器，先把被替换的 current record 保存为 revision，再原子更新当前 record。默认档位为 `full`，并启用优化、频率和 TDDFT 解析；`--no-opt`、`--no-freq`、`--no-tddft` 可分别关闭对应部分。`--profile` 是 `--result-profile` 的别名。

6.17.3.7 创建 case / project 归档

```
btdb case|project archive create --out <file.btdb> [--path <path>]
    [--artifacts none|minimal|standard|full]
    [--include-native-wavefunctions] [--max-artifact-bytes N]
    [--no-recipes] [--no-published-results] [--no-workflow-context]
    [--overwrite]
```

创建前会执行全量同步和完整预检。归档采用不可变 ZIP64 容器，包含 current records、revisions、由记录可达的 SHA-256 对象，以及按选项纳入的内容：

- 默认包含 `Results/Recipes`；`--no-recipes` 排除它；
- 默认包含 `Results/published.json` 显式选择的文件；`--no-published-results` 排除它们；
- 默认包含可用的顶层 `.bt` / `.projbt` 等工作流上下文；`--no-workflow-context` 排除它们；
- SQLite 索引、快照、队列状态和其他运行期临时状态不进入归档；
- 默认不覆盖已有 `.btdb`；`--overwrite` 只对 `restore` 和 `archive create` 有效。

三个 `--no-*` 归档选项只允许用于 `archive create`。

6.17.3.8 顶层 `.btdb` 归档命令

命令或选项	行为
`archive info`	读取 `btdb.json` 并显示 archive ID、作用域、创建时间、producer、内容根哈希和条目统计，不解包 payload。
`archive verify`	校验 ZIP CRC、manifest 条目大小、SHA-256 和 manifest content root。
`archive restore`	把归档恢复到新目录。
`archive import`	`archive restore` 的整体恢复别名；不是把记录合并到现有数据库。
`<file.btdb>`、`--path <file>`、`--archive <file>`	三种等价的归档路径写法。
`--out <dir>`	`restore` / `import` 必需的目标目录。

命令或选项	行为
`--overwrite`	允许替换目标目录中的已有文件。
`--no-rebuild`	只解出持久事实，不重建 `.bane/cache/project.db`。
`--materialize`	重建索引后，额外把已登记工件恢复到 `Recovered/`。不能与 `--no-rebuild` 同用。

`info` / `verify` 不接受恢复专用选项。

6.17.3.9 报告模板

```
btdb case|project template list [--scope any|case|project]
btdb case|project template show <name>
btdb case|project template query <name> [--path <path>]
    [--param key=value|-P key=value]... [key=value ...]
    [--format table|tsv|csv|json|markdown] [--out <file>]
btdb case|project template export <name> ...
```

模板名大小写不敏感，连字符与下划线会归一化。`--param` / `-P` 可重复；模板名后的裸 `key=value` 也会作为参数。重复提供同一 key 时，后值覆盖前值。`query` 通常写标准输出；指定 `--out` 后也可写文件。`export` 与 `query` 使用同一解析和同步流程。

内置模板如下：

- `latest-overview`：case / project；参数 `case_key`、`task_name`、`template_task_name`、`limit`。
- `case-report`：case / project；参数 `case_key`。
- `task-report`：case / project；参数 `task_name`、`case_key`。
- `template-family`：case / project；参数 `template_task_name`。
- `matrix-slice`：case / project；参数 `matrix_key`、`matrix_value`、`template_task_name`。
- `artifact-inventory`：case / project；参数 `case_key`、`task_name`。
- `compare-pair`：case / project；参数 `left_task`、`right_task`、`metric`、`case_key`；数值字段会计算 `delta`。
- `recent-ingest`：case / project；参数 `limit`。

`template show <name>` 显示标题、作用域、默认格式、参数定义、SQL 和示例。

6.17.3.10 Obsidian 导出

```
btdb case|project obsidian [export] [--path <path>] [--out <dir>]
    [--paths none|relative|absolute|collapsed]
    [--dataview-from <scope>]
```

导出前先同步数据库。默认输出到 `<root>/Results/Obsidian`，生成 `main.md`、`tasks/`、`structures/`、`data/` 和 `tables/`；project 作用域还生成 `index.md`。

- `--paths` 控制正文中的复现路径：`none` 隐藏，`relative` 使用相对路径，`absolute` 使用绝对路径，`collapsed` 折叠显示；默认 `collapsed`。
- `--dataview-from` 设置生成查询的 Dataview `FROM` 范围。以 `#`、`"` 或 `[` 开头的值原样使

用；其他值按 vault 文件夹路径加引号。默认 **#btdb**。

6.17.3.11 任务指标比较

```
btdb compare <left-task> <right-task>
    [--metric <key>] [--path <project-dir|file.projbt>]
    [--format table|tsv|csv|json|markdown] [--out <file>]
```

默认 metric 为 **energy**。metric key 只能包含字母、数字、下划线、点和连字符；固定列直接查询，其他 key 从 **values_json** 读取。**--path** 也可作为两个任务名后的一个位置参数。默认输出为终端表格；给出 **--out** 后写入文件。

6.17.3.12 校验与退出状态

- **verify** 重新计算数据库中每个已登记对象的 SHA-256 并校验字节数；它不会先同步、回填或修复对象。需要纳入新 records 时先运行 **sync**。
- 命令行或查询参数错误返回非零；同步、恢复、重新解析、归档或校验出现部分失败时也返回非零。脚本不应只解析终端文字，应同时检查退出状态。

6.17.3.13 示例

```
btdb case sync --path ./caseA --artifacts standard
btdb case verify --path ./caseA
btdb case restore --path ./caseA --run-id opt_caseA --artifact output.primary
 ↪  --out ./restored
btdb case reparse --path ./caseA --run-id opt_caseA --result-profile full
btdb case query --path ./caseA --sql "SELECT task_name, energy FROM
 ↪  task_flat_latest"

btdb project template show compare-pair
btdb project template export compare-pair --path ./demo_project \
  -P left_task=opt -P right_task=td --format markdown --out compare.md
btdb project obsidian export --path ./demo_project --out ./vault/demo_project \
  --paths collapsed --dataview-from Projects/demo_project
btdb compare opt td --metric energy --path ./demo_project

btdb project archive create --path ./demo_project --out demo_project.btdb
btdb archive info demo_project.btdb
btdb archive verify --archive demo_project.btdb
btdb archive restore demo_project.btdb --out ./restored_project --materialize
```

6.17.4 btool

btool 处理模板、project 骨架、批量 case 操作、任务块改名、DSL v2 格式化/迁移/展开、结构解释和静态检查。数据库同步、查询、报告和归档由 **btdb** 负责。

6.17.4.1 命令总览

```
btool <command> [options]
btool help <command>
```

```
btool template <list|categories|merge|variable|interactive> ...
btool project <init|add|update|rebuild|exec|collect> ...
btool rename <task.bt|task.projbt> <old> <new>
btool fmt <task.bt> [-o FILE|--in-place]
btool migrate <task.bt> [-o FILE|--in-place]
btool expand <task.bt> [-o FILE] [--expanded]
btool explain <task.bt> [--task NAME] [--format text|json] [--expanded]
btool lint [--strict] [--quiet] [--format text|json] [PATH...]
```

顶层命令和 **project** 子命令都接受无歧义前缀，例如 **btool proj co ...** 等价于 **btool project collect ...**。前缀同时匹配多个命令时会报错，不执行猜测。

6.17.4.2 模板命令

```
btool template list [-d|--dir <template-dir>]
btool template categories|category [-d|--dir <template-dir>]
btool template merge [-o|--output <file|->] [-d|--dir <template-dir>]
↪  <template...>
btool template variable <task.bt> <var-template>
btool template interactive
```

- **list** 列出模板；**categories** / **category** 按类别分组。
- **-d** / **--dir** 覆盖模板目录。省略时使用 **BANETASK_TEMPLATE_PATH**，再回退到 **./conf/bt_templates**。
- **merge** 至少需要一个模板名。默认写 **merged_task.bt**；**-o -** 写标准输出。模板不存在或输出写入失败时返回非零。
- **variable** 根据指定任务文件生成变量定义模板。
- **interactive** 交互选择并合并模板。预览和最终 **.bt** 文件头会包含且只包含一条 **autorun: true**；模板中的顶层 **autorun** 统一规范为 **true**；该值等价于 **default**，因此 preprocess 和主命令成功后允许继续触发 **btrun kick**。

顶层别名如下：

别名	等价命令
btool -l, **btool --list**	**btool template list**
btool -c, **btool --category**	**btool template categories**
btool -m, **btool --merge**	**btool template merge**
btool -v, **btool --variable**	**btool template variable**
btool -i, **btool --interactive**	**btool template interactive**
btool -t, **btool --test**	在当前目录创建 **test/test.bt** 与 **test/test.xyz** 测试文件。

6.17.4.3 project 命令

```
btool project init
btool project add [--path PATH] [--dry-run] <xyz...>
btool project update [--path PATH] [-n|--name TASKS] [--run-comd] [--dry-run]
btool project rebuild [--path PATH] [-n|--name TASKS]
↪  [--run-script|--run-runner] [--dry-run]
btool project exec [--path PROJECT] -n TASKS [--dry-run] -- <command...>
btool project exec [--path PROJECT] -n TASKS [--dry-run] --cmd "<command>"
btool project collect [--path PROJECT] -n TASKS [--dry-run] <glob...>
```

6.17.4.3.1 路径与任务选择

- **add**、**exec**、**collect** 的 **--path** 接受 project 根目录或 **.projbt**。
- **update**、**rebuild** 还接受单个 case 目录或 case **.bt**，此时只处理该 case。
- 省略 **--path** 时，优先使用当前目录中唯一的 **.projbt**。
- **-n** / **--name** 可重复，也可用逗号分隔：**-n opt -n soc** 与 **-n opt,soc** 等价。重复名称会去重。
- 任务名支持 shell 风格通配，例如 **-n 'opt*'**、**-n '*'**。通配符应加引号，避免被调用 shell 提前展开。
- **update** / **rebuild** 省略 **-n** 时进入交互式任务选择；**exec** / **collect** 必须显式给出 **-n**。
- **--dry-run** 只打印计划，不修改文件或执行命令。

6.17.4.3.2 project init

project init 在当前目录交互创建 project：

1. 读取 **BANETASK_TASK_PATH** 作为目标根目录，提示 project name；
2. 从当前目录选择 **.bt** / **.projbt** 模板；
3. 扫描 **.xyz**、**.gjf**、**.com**、**.inp**、**.log**、**.gms** 几何文件；
4. 创建 project 目录、**<project>.projbt**、各 case 目录和 case 入口；
5. 原样复制几何文件；case **.bt** 使用指向主 **.projbt** 的符号链接，不能创建符号链接时使用 **&INCLUDE** stub；
6. 可选择立即运行 **btrun kick --path <project-dir>**。**btrun** 路径优先读取 **BANETASK_BTRUN_PATH**。

btool -b / **btool --batch** 是同一交互流程的别名。

6.17.4.3.3 project add

add 向现有 project 增量导入一个或多个 **.xyz**。它为每个 xyz 创建同 stem case、复制结构并接入主 **.projbt**；已存在的 case 目录会跳过。该命令不接受 **-n**。需要从 **.gjf/.com/.inp/.log/.gms** 初始化项目时使用 **project init**。

6.17.4.3.4 project update

update 对选定任务执行 **banetask --comd-only**，只刷新 **pre_comd** / **comd**，不完整重建任务。

- **--run-comd** 在刷新后立即执行每个选定任务的 **comd**；
- 未指定 **--run-comd** 时，交互流程可选择立即执行或稍后执行；稍后执行会在解析出的 project 根或 case 目录写 **btool_update_comd.sh**，按顺序复现同一组 **bash comd**；
- **btool -u** / **btool --update** 是别名，并接受路径、**-n**、**--run-comd** 和 **--dry-run**。

6.17.4.3.5 `project rebuild`

`rebuild` 清理并完整重建选定任务，不使用 **`--comd-only`**。**`--run-script`** 在 project 级重建完成后调用配置的 **`btrun kick`**；**`--run-runner`** 是其别名。单 case 路径不允许使用该启动步骤：先重建 case，再从 project 根运行 runner。

6.17.4.3.6 `project exec` 与 `project collect`

- **`exec`** 在每个选定任务目录执行同一条 shell 命令。命令可写在 **`--`** 后，也可通过 **`--cmd`** 作为一个字符串传入。
- **`collect`** 从每个选定任务目录收集一个或多个 glob 匹配文件。
- 两者都要求显式任务选择。**`--exec`**、**`--collect`**、**`--rebuild`** 三个操作别名互斥。

仍接受 file-first 别名：

```
btool demo.projbt -n opt --rebuild --run-script
btool demo.projbt -n soc --exec 'cat soc*.log'
btool demo.projbt -n soc --collect '*.log' '*.png'
```

它们分别等价于相应的 **`btool project rebuild|exec|collect --path demo.projbt ...`**。新脚本宜使用显式子命令。

6.17.4.4 任务块改名

```
btool rename <task.bt|task.projbt> <old-block> <new-block>
```

- 对 **`.bt`**，同步更新任务文件、**`bt.status`**、任务目录、相关文件名前缀、依赖该块名的 **`.bane`** 元数据/记录和 **`Results/`** 派生工件。
- 对 **`.projbt`**，更新项目模板，并在各 case 中同步任务目录和派生工件。
- 内容寻址对象目录不会做文本替换；派生 SQLite 索引会被删除，下一次 **`btdb sync/rebuild`** 从改名后的记录重建。
- **`.bt`** 若是指向 **`.projbt`** 的符号链接，会自动按 project 模式处理。

别名：

```
btool caseA.bt -r opt opt2
btool caseA.bt --rename opt opt2
```

6.17.4.5 格式化与迁移

```
btool fmt <task.bt> [-o|--out|--output <file> | -i|--in-place]
btool migrate <task.bt> [-o|--out|--output <file> | -i|--in-place]
```

- **`fmt`** 用共享 parser 读取当前 DSL，输出 canonical DSL v2，只调整布局，不改变语义。
- **`migrate`** 读取受支持的输入别名，并输出 canonical DSL v2。
- **`-o`** / **`--out`** / **`--output`** 写新文件；**`-i`** / **`--in-place`** 覆盖输入；两者不能同时使用。
- 两种输出选项都省略时，结果写标准输出，输入文件不变。

6.17.4.6 展开模块化任务文件

```
btool expand <task.bt> [-o|--out|--output <file>]
↪  [--expanded|--matrix-expanded]
```

`expand` 解析 YAML/task include、上下文片段、**`.btlib`** 命名片段和 **`@foreach`**，输出 canon-

ical DSL v2：

- 默认保留 `matrix:` 声明和 matrix 展开前的逻辑任务；
- `--expanded` / `--matrix-expanded` 输出最终 matrix 任务并移除 `matrix:`；
- 未给输出文件时写标准输出；
- 输出路径不得与输入文件相同，命令从不原地改写模块化源文件。

6.17.4.7 解释解析结果

```
btool explain <task.bt> [--task <name>] [--format text|json] [--expanded]
```

`explain` 输出任务块、指令、源跨度和展开结构。`--task` 只解释一个任务；`--expanded` 使用 matrix / `@foreach` 展开后的任务名；默认格式为 `text`，`json` 适合工具消费。声明 `%control finalgeom` 时，两种格式都包含 `finalgeom`，可核对下游 `%source` 将读取的最终几何文件。

6.17.4.8 静态检查

```
btool lint [--strict] [--quiet] [--format text|json] [<file.bt|dir>...]
```

- 文件参数只处理 `.bt`；目录会递归扫描其中的 `.bt`。`.projbt` 不作为 lint 输入。
- 省略路径时递归扫描当前目录。
- 检查使用与运行时相同的 parser AST 和源跨度。除语法错误外，还检查任务结构、失效 `%source` / `%when` / `guess` 引用、`other` 缺少 `%template`、`script` 缺少 `%body`，以及可迁移到 DSL v2 的输入别名。
- `--strict` 把 warning 按 error 计入汇总和退出状态。
- `--quiet` 只在 `text` 模式隐藏逐条诊断；汇总仍输出，错误退出状态不变。
- `--format text` 为默认人读格式；`--format json` 输出完整文件列表、诊断和 summary，`--quiet` 不删减 JSON。
- 存在有效 error 时返回 `1`。所有输入都不存在，或递归扫描后没有找到 `.bt` 文件时，向标准错误输出 `no .bt files found` 并返回 `1`。

6.17.4.9 示例

```
btool template merge opt freq -o task.bt
btool template merge -d ./conf/bt_templates opt freq -o -

btool project add --path demo.projbt mol1.xyz mol2.xyz
btool project update --path ./demo_project -n opt,soc --run-comd
btool project rebuild --path demo.projbt -n opt,soc --run-script
btool project exec --path demo.projbt -n 'opt*' -- 'grep Normal *.log'
btool project collect --path ./demo_project -n soc 'plot/*.png' '*.txt'

btool rename demo.projbt old_block new_block
btool fmt task.bt > formatted.bt
btool migrate old.bt --in-place
btool expand task.bt --expanded -o flattened.bt
btool explain task.bt --task opt --expanded --format json
btool lint --strict ./cases
```

6.17.5 btc

btc 用于把 **.btc** 模板转换为单 case **.bt** 或批量 **.projbt** 项目布局。它更像是一个“项目初始化器”：既可以对着一份模板快速起一个 case，也可以把一批结构文件一次性铺成标准 project 骨架。

6.17.5.1 用法

```
btc <template.btc> [--geom|--xyz <file|glob>]... [file|glob ...]
    [--project-dir|--project <dir|file.projbt>]
    [--script|--runner|--runsh|--runbat <path>]
    [--no-run] [--detach]
btc -h|--help
```

--xyz 是 **--geom** 的别名；**--runner**、**--runsh**、**--runbat** 是 **--script** 的别名；**--project** 是 **--project-dir** 的别名。

6.17.5.2 模式与 geometry 规则

btc 接受 **.xyz**、**.gjf**、**.com** geometry，并按单 case 或 project 两种模式运行：

- 未显式给出 geometry，或最终只有一个显式 geometry 时，**btc** 保持单 case 模式。
- 只有**显式**传入两个及以上 geometry（位置参数、重复 **--geom** 或别名 **--xyz**）时，才会切换到 **.projbt** project 布局；自动探测到多个文件仍会报错并要求显式指定。
- 自动探测先查当前目录，再查模板 **.btc** 所在目录。
- 在同一目录中，若存在 **.xyz**，自动探测只使用 **.xyz**；只有完全没有 **.xyz** 时，才回落到 **.gjf/.com**。
- 位置参数既可以由当前 shell 预先展开为多个文件，也可以直接传入带通配符的模式字符串；展开结果会按规范路径去重。
- 若多份显式 geometry 最终具有相同 stem（例如 **a.xyz** 与 **a.gjf**），**btc** 会直接报错，避免 project 模式下 case 目录冲突。
- 单 case 且显式传入单个 geometry 文件时，输出 **.bt** 会写到该 geometry 所在目录。
- project 模式若未显式给出 **--project-dir** / **--project**，默认输出目录为 **<template_stem>/**，主入口文件为 **<template_stem>.projbt**。若参数本身以 **.projbt** 结尾，则视为显式项目入口文件路径。
- project 布局会把选定几何文件原样复制进各 case 目录，并为每个 case 创建一个指向 **.projbt** 的 **.bt** 入口。

6.17.5.3 ask: 交互、define: 回写与变量模板

.btc 模板可选写 **ask:** 块，用于声明需要在生成阶段交互填写的变量。**btc** 会按 **ask:** 中列出的变量发起交互。例如：

```
ask:
  func: b3lyp
  basis
  scrf: sdd

define:
  basis: def2-svp
```

规则如下：

- `ask:` 中每项既可以写成 `name`，也可以写成 `name: default`。
- 若模板里已有 `define:`，对应值会被视为当前值，并优先于 `ask:` 默认值。
- 非 TTY 顺序输入模式下，输入 `r` / `retain` 会保留当前值；若没有当前值但有 `ask:` 默认值，则保留默认值。
- 当标准输入是 TTY 时，`btc` 会进入编号菜单；也可以通过 `BTC_FORCE_ASK_MENU=1` 强制启用该交互菜单。
- 交互菜单支持从变量模板目录递归加载 `.yaml` / `.yml` 模板。目录优先取 `BANETASK_TEMPLATE_PATH`，否则回落到 `~/.bane/task/bt_templates`。
- 变量模板采用 `name:` / `description:` / `variables:` 结构，其中 `variables:` 里只有与当前 `ask:` 同名的键才会被应用。
- 交互完成后，新增或修改过的值会被合并回 `define:`；若原模板没有 `define:` 块，则会在文件顶部自动补出新的 `define:`。

6.17.5.4 runner 与输出规则

`--script` / `--runner` / `--runsh` / `--runbat` 可显式指定后续调用的 runner。runner 可以是 `btrun`、`run.sh` 或 `run.bat`，查找顺序如下：

1. 显式 `--script` / `--runner` / `--runsh` / `--runbat`
2. `BANETASK_BTRUN_PATH`
3. `BANETASK_SCRIPTS_PATH` 下的 `btrun`（目录级候选还会回退到 `run.sh` / `run.bat`）
4. `~/.bane/task/scripts/`、`PATH` 与 `btc` 可执行文件同目录下的 `btrun`；随后检查 `~/.banetask/scripts/`、`run.sh` 和 `run.bat`。
5. 变量别名 `BANETASK_RUNSCRIPT` / `BANETASK_RUN_SCRIPT` / `BANETASK_RUNBAT` / `BANETASK_RUN_BAT` / `BANETASK_RUNSH` / `BANETASK_RUN_SH`

调用方式也分两类：

- 若解析到的是 `btrun`，`btc` 会调用 `btrun kick --path <directory>`。
- 若解析到的是 `run.sh` / `run.bat`，则调用 `<runner> -p <directory>`。

补充说明：

- `--no-run` 只生成文件，跳过 runner 调用。
- `--detach` 以分离方式启动 runner，并立即返回。
- 若目标 `.bt` / `.projbt` 已存在，覆盖前会自动备份为带时间戳的 `.bak...` 文件。
- project 模式会优先为各 case 创建指向 `.projbt` 的符号链接；若平台或文件系统不支持符号链接，则自动回退为 `&INCLUDE <relative projbt>` stub。

6.17.5.5 示例

```
btc template.btc
btc template.btc mol.xyz
btc template.btc mol.gjf
btc template.btc --geom mol.com
btc template.btc *.xyz
btc template.btc --geom a.gjf --geom b.com --project-dir batch_opt
btc template.btc --project batch_opt/batch_opt.projbt a.xyz b.xyz --no-run
btc template.btc --script ~/.bane/task/scripts/btrun --detach
```

6.17.6 btproc

btproc 汇集结果提取、轨迹收集、逐原子向量、FCclasses 报告和方法学说明等结果处理命令。以下条目按公开命令组组织。

6.17.6.1 btproc fork collect

btproc fork collect 用于把多帧 XYZ 或程序输出中的几何轨迹收集成 **extra.yaml** 中的结构化集合，供 **@foreach?**、**%source xyz=<name> frame=<N>** 和 **%when** 使用。

```
btproc fork collect <xyz|gaussian|orca|gms|amesp|bdf|mrcc|maple|mopac|auto>
↪  <input> --as <collection> [--out <extra.yaml>] [--conv] [--xyzmeta]
↪  [--xyzmeta-as <alias>] [--xyzmeta-field <source> <alias>] [--const <alias>
↪  <value>] [--dist <i> <j> <alias>] [--angle <i> <j> <k> <alias>]
↪  [--dihedral <i> <j> <k> <l> <alias>]
```

补充说明：

- **xyz** 直接读取多帧 XYZ。
- **gaussian**、**orca**、**gms**、**amesp**、**bdf**、**mrcc**、**maple**、**mopac** 会先从程序输出中抽取几何轨迹，再归一化为内部 XYZ 轨迹后写入集合。
- **auto** 用于让工具按输入文件特征自行判断解析路径。
- **--conv** 用于仅保留已收敛的结构帧；该选项面向支持优化步收敛标记的程序输出。若某后端没有逐步收敛标记，工具保留其可用的最终结构。
- 默认输出文件为 **extra.yaml**。
- 若目标 YAML 已存在，工具会保留其他顶层键，并覆盖同名集合。
- 生成的集合元素至少包含 **index** 与 **frame** 两个 1-based 字段。
- **--xyzmeta** 会写入可获得的 XYZ frame metadata；配合 **--xyzmeta-as <alias>** 时会把这些字段放进指定子映射。
- 可重复追加 **--xyzmeta-field <source> <alias>** 以只选择部分 metadata 字段并重命名；常用 source 包括 **comment**、**energy**、**maxForce**、**rmsForce**、**maxDisp**、**rmsDisp**、**charge**、**multiplicity**，以及 XYZ 注释行中的 **key=value** 字段。
- 可重复追加 **--const <alias> <value>**，把同一个常量字段写入每一帧记录；数值字符串会按数字写入，其他值按字符串写入。
- 可重复追加 **--dist** / **--angle** / **--dihedral**，把距离、键角、二面角字段写入每一帧记录。

示例：

```
btproc fork collect xyz cluster.xyz --as cluster --xyzmeta --xyzmeta-as meta
↪  --const method "B3LYP-D3 BJ" --const temperature 298.15 --dist 1 2 r12
↪  --angle 3 4 5 a345 --dihedral 6 7 8 9 d6789 --out extra.yaml
btproc fork collect xyz cluster.xyz --as cluster --xyzmeta --xyzmeta-field
↪  energy E --xyzmeta-field Boltzmann_w w --xyzmeta-field Cum Cum --out
↪  extra.yaml
btproc fork collect gaussian opt.log --as optscan --conv --xyzmeta
↪  --xyzmeta-field energy E --dist 1 2 r12 --out extra.yaml
btproc fork collect mopac opt.out --as mopac_opt --conv --xyzmeta
↪  --xyzmeta-field energy E --out extra.yaml
```

运行后即可在任务文件中写：

```
@foreach? cluster
$opt_[cluster.index]
  %source xyz=cluster frame=[cluster.frame]
  %keywords "opt b3lyp/6-31g*"
@end
```

6.17.6.2 btproc result

btproc result 用于从日志中提取结果，并写出 JSON、快照和数据库记录。它处在结果链路的核心位置，既服务于 **%when** 所需的快照补建，也服务于数据库同步所需的规范化记录生成。

```
btproc result extract < 输入日志文件 > [输出 JSON 文件] [options]
btproc result extxyz-kv <file> [--frame first|last|N]
btproc result list-properties [--json]
btproc result describe-property <name> [--json]
btproc result query-property <output-file> <name> [--unit <unit>] [--json]
```

当前主要选项如下：

- **-o**, **--opt**：提取优化/收敛信息。
- **-f**, **--freq**：提取频率信息。
- **-td**, **--tddft**：提取激发态、吸收峰和自旋轨道耦合（SOC）信息。
- **--profile <summary|standard|full>**：选择规范结果档位。**summary** 只保留最终结构与摘要；**standard**（默认）保留轨迹和所选分析，但不复制原始日志；**full** 额外把完整源日志嵌入规范文档，适合审计和离线归档。
- **--meta <file>**：合并 banetask 侧边元数据 JSON。
- **--extra-kv <file>**：合并用户补充结果 kv，并同时进入快照与记录。
- **--record-kv <file>**：合并 record-only 补充 kv。
- **--values-kv <file>**：写入运行期 values kv，包含 **atoms.N.*** 等细粒度字段。
- **--extxyz-kv <file>**：从 extxyz per-atom 属性输出 kv 行。
- **--frame <first|last|N>**：配合 **--extxyz-kv** 选择帧；数字从 1 开始，默认使用最后一帧。
- **--db-kv <file>**：写入轻量级 kv 快照。
- **--db-snapshot-copy <file>**：额外写一份集中式快照副本。
- **--db-record <file>**：写入规范化记录 JSON。
- **--db-ready <file>**：写入 ready 标记文件。
- **--list-properties**：列出统一 QM 属性，包括 shape、物理维度、规范单位和序列轴。
- **--describe-property <name>**：说明一个规范属性或别名，并显示所需解析内容。
- **--query-property <file> <name>**：直接从量化输出查询带类型的标量或序列；可用 **--unit** 转换兼容单位。

补充说明：

- 第一个位置参数是输入日志文件路径；第二个位置参数才是可选的 JSON 输出文件。
- 只要使用任一 **--db-kv**、**--values-kv**、**--db-snapshot-copy**、**--db-record** 或 **--db-ready**，程序就会自动启用 **-o**、**-f** 和 **-td** 三类提取。
- **--record-kv** 适合写入只需要进入标准记录 envelope、但不希望污染轻量级快照命名空间的字段，例如原子级、片段级或额外分析结果。**--values-kv** 会把最终 payload 中的可扁平化 scalar values 写成运行期 kv，供后续任务读取。

- `--extxyz-kv` 只做 extxyz 到 kv 的转换并写到标准输出；常见用法是在 `dbcollect --extxyz` 中把这些 kv 行转入 `.bane.dbcollect.user.kv`。它会跳过 `species`、`z`、`pos` 等核心列；三维向量属性会展开为 `.x`、`.y`、`.z`，其他多列属性会展开为 `.1`、`.2` 等后缀。
- 启用上述 DB 输出且省略 JSON 输出文件时，标准输出保持清洁，便于嵌入脚本流程。
- 写入 `--db-record` 时，`btproc result` 会按 `BANETASK_DB_ARTIFACT_PROFILE` 等数据库工件策略归档可发现文件；若目标 record 已存在，则先把被替换的 envelope 写入 `revisions/`，再原子性更新当前记录。
- 可用的结果提取后端取决于当前安装版本是否包含 Gaussian / GAMESS / ORCA / AMESP / MOPAC / BDF 等结果解析能力。MOPAC 解析会在主 `.out` 旁自动合并同 stem 的 `.aux`。
- 属性发现、直接查询、Plot DSL 的 `result(task, property)` 使用同一套属性名。统一属性和缺失状态见本手册“统一量化结果查询”章节。

6.17.6.3 `btproc atomvec`

`btproc atomvec eval` 是 `atomvec` 命令 DSL 的独立入口。它从任务目录读取 atom-level values，计算目标向量，写入运行期 kv、record-only kv 和最小结果快照，并可导出 extxyz 或 CHG。

6.17.6.3.1 用法

```
btproc atomvec eval --dsl '<target> = <expr> [dsl-options]'
    [--values-out <file>] [--record-kv <file>] [--snapshot-kv <file>]

btproc atomvec eval '<target> = <expr> [dsl-options]'
    [--values-out <file>] [--record-kv <file>] [--snapshot-kv <file>]
```

未给 `--dsl` 时，剩余参数按空格连接为 DSL；同时给出两种形式时使用 `--dsl`。表达式为空或不含顶层 `=` 时返回非零。

6.17.6.3.2 文件输出选项

选项	默认值与行为
`--dsl <expr>`	指定完整 DSL。
`--values-out <file>`	默认 `.bane.values.kv`；追加 `atoms.N.<target>=<value>`。
`--record-kv <file>`	默认不写；给出后把同一组 atom values 追加到 record-only kv，常用 `.bane.dbcollect.user.kv`。
`--snapshot-kv <file>`	默认不写；给出后写最小结果快照。

最小快照包含 `schema_version=1`、`status=done`、`atom_count` 和 `geometry.atom_count`。环境中存在 `BANETASK_TASKNAME`、`BANETASK_INPUTNAME`、`BANETASK_STRUCID` 时，也会写入相应任务标识。

6.17.6.3.3 DSL 选项

独立入口与 `%preprocess` / `%process` 中的 `atomvec` 使用同一语法。向量引用、函数和标量表达式见本手册“Process DSL / `atomvec`”一节。可用选项为：

选项	行为
`--check same_atoms`	校验向量长度；可读取几何时同时校验元素序列。
`--sum <scalar>`	求值后把向量平移归一化到指定总和。
`--renorm <scalar>`, `--renormalize <scalar>`	`--sum` 的别名。
`--out <file>`, `--extxyz <file>`	使用所选几何写 extxyz。
`--chg <file>`	使用同一几何写 CHG 文本，每行包含元素、坐标和派生值。
`--format extxyz`	extxyz 输出格式；只接受 `extxyz`。
`--geom <task>`	指定 extxyz / CHG 的几何来源。省略、写 `first` 或 `auto` 时使用表达式中的第一个向量来源；没有向量来源时使用当前目录。

`--out` 与 `--chg` 可同时使用。几何原子数必须与结果向量长度一致；找不到几何时不写部分输出并返回非零。

6.17.6.3.4 示例

```
btproc atomvec eval --dsl \
  "charge.avg = mean(task_a:charge.resp, task_b:charge.resp) --check same_atoms
↪  --sum @task_a.charge --out charge_avg.extxyz --chg charge_avg.chg" \
  --values-out .bane.values.kv \
  --record-kv .bane.dbcollect.user.kv \
  --snapshot-kv .bane.result.kv
```

任务文件中的等价写法：

```
%process
  atomvec charge.avg = mean(task_a:charge.resp, task_b:charge.resp) --check
↪  same_atoms --out charge_avg.extxyz --chg charge_avg.chg
```

6.17.6.4 `btproc method`

`btproc method` 用于根据 BaneTask 任务文件、任务工件清单、已生成输入文件和内置方法参考库生成“计算方法”说明。它既可以手动运行，也会在未显式覆盖动作集的默认 `finalize` 尾部自动运行；也可以通过 `finalize.method.generate` 显式启用并配置输出路径、参考库和严格模式。

```
btproc method generate --path <case_or_project_root> --task-file <file.bt>
↪  [options]
btproc method refs lint [--refs <file>]
```

`generate` 常用选项如下：

- `--path <dir>`：case 或 project 根目录；用于查找任务目录、manifest 与输出目标。
- `--task-file <file>`：用于枚举任务的 `.bt` / `.projbt` 文件。
- `--origin <name>`：指定 origin basename；未给出时默认使用任务文件 stem。

- `--out <file>`：Markdown 输出；默认 `Results/Method/method.md`。
- `--json <file>`：JSON 输出；默认 `Results/Method/method.json`。
- `--refs <file>`：方法参考库；未给出时依次读取环境变量 `BTMETHOD_REFS_PATH`、`banetask.conf` 中的 `BTMETHOD_REFS_PATH`、安装默认路径、`~/.bane/task/references.db`，以及当前目录下的 `share/banetask/method/references.db`。
- `--scope <scope>`：作用域上下文标签，默认 `case`。`case` 与 `project` 使用相同的事实采集和文档生成规则。
- `--strict`：遇到高风险 warning（如缺少输入、路线解析失败、参考条目缺失）时返回非零；默认关闭。
- `--no-strict`：显式关闭严格模式，适合覆盖 `finalize` 或包装脚本传入的严格设置。

6.17.6.4.1 方法事实来源

方法生成只读取当前任务计划中未被 `%when` 屏蔽的有效任务。每个任务必须已有任务目录；随后按以下顺序取得方法事实：

1. 若 `task.manifest.json` 列出了 Gaussian、ORCA 或 AMESP 输入文件，逐个解析所有存在的输入文件。一个 manifest 中的多个输入不会只取第一个；它们分别进入方法工作流。
2. Gaussian 输入含 `--Link1--` 时，每个 job 作为独立计算阶段处理。后续 job 使用 `Geom=AllCheck` 等方式省略电荷和多重度时，会沿用前一阶段的显式值。
3. 没有可用输入事实时，才回退到任务文件中的 `%keywords`、程序额外输入块或 `%args`。回退来源会写入 JSON provenance，并在标准错误中给出 notice。
4. XTB 没有独立路线输入文件时，优先读取 manifest 的 `program_params.args`，否则读取任务 `%args`。

实际输入存在但解析失败时，即使任务文件回退仍可生成内容，`--strict` 也会把该解析失败视为错误。manifest 中列出的缺失输入文件同样属于严格模式错误。

6.17.6.4.2 BaneWfn 引用与分析汇总

对每个有效任务，方法生成器还会检查任务目录中的 `references.bib`。文件可以由同一任务或多个任务中的 BaneWfn 调用生成，生成器会按规范化 DOI 优先、随后按 `baneid` 或 BibTeX key 合并，保持第一次出现的引用顺序。

BaneWfn 扩展字段的约定如下：

- `baneid`：稳定的引用标识；其中 `banewfn` 作为 BaneWfn 前端软件引用，其余条目作为 Multiwfn 分析引擎及方法引用。
- `banetext`：可直接写入方法参考文献列表的完整文本；缺少时由标准 BibTeX 字段生成。
- `banereason1`、`banereason2` 等：当前引用对应的分析名称。空值忽略，重复值去重。已包含 `analysis` / `analyses` 的文本保持原样；`IGMH`、`ESP` 等已知短名称自动补 `analysis`，其他自定义文本按原样使用。

所有非空分析名称会合并成一句方法描述，并在 `Multiwfn` 后开启引擎与方法文献编号，在 `banewfn` 后单独标注前端文献编号。例如：

```
AAA, energy potential analysis(ESP) and IGMH analysis were performed using
↪  Multiwfn[1-4] and banewfn[5].
```

当 BibTeX 中存在引用但没有非空 `banereasonN` 时，使用通用名称 `Wavefunction analysis`。格式错误的 BibTeX 会产生 parse warning，并在 `--strict` 模式下使生成失败。导

入的引用正文、句内引用分组及来源文件会同步写入 `method.json`，因此 `finalize.method.generate` 无需额外动作即可获得相同结果。

6.17.6.4.3 计算步骤推断

生成器优先采用输入中显式声明的计算类型：

- 优化、单点能、频率、梯度、垂直激发和分子动力学按输入阶段生成对应步骤。
- 同一阶段同时声明 TD-DFT/TDA 与 `Opt`、`Freq` 或 `Force` 时，激发态理论、态数和 root 作为结构操作属性保留；`method.md` 将该阶段归并为一条 TD-DFT 方法说明，不另外描述激发态几何优化。独立 TD-DFT/TDA 阶段写出显式理论级别。sTDA 仍按独立激发态方法步骤处理。
- 仅出现 method/basis，且没有任何显式 job 类型时，按程序默认行为生成单点能步骤。
- 频率、梯度、TD-DFT/TDA/sTDA 或分子动力学任务不会仅因存在 method/basis 而附带额外单点能步骤。
- AMESP 的 `force` / `numforce` 按解析/数值梯度描述，不作为普通单点能；`freq` / `numfreq` 分别标记解析/数值频率。
- XTB 使用参数 token 判断 `--opt`、`--ohess`、`--hess`、`--grad`、`--md` 等操作；文件名或路径中的 `opt`、`freq` 字样不会触发任务类型。
- 任务块的 `%batch` 中存在 `cluster` 时，该任务直接作为构象搜索步骤进入方法工作流。若其 `%source` 直接依赖分子动力学任务，生成器将该动力学过程描述为构象采样轨迹的来源；若直接使用 `origin`，则按预先准备的构象集合描述。
- `cluster dist` 按原子间距离矩阵聚类描述，`cluster rmsd` 按刚体叠合 RMSD 聚类描述。正文保留相应几何阈值、能量阈值及 `min` / `accum` Boltzmann 筛选参数，不根据聚类结果继续推断后续正式计算用途。

DFT 方法正文使用 `DFT Calculations` 章节，并以“Ground state geometries were optimized using Density Functional Theory (DFT) …”描述基态优化；MP2、耦合簇、多参考、半经验或复合方法使用中性优化表述并保留实际方法名称。Gaussian 任务由软件总述句统一写为 `Gaussian 16` 并绑定 Gaussian 16 程序参考文献；ORCA、AMESP 与 xTB 未取得可靠版本信息时只写程序名。

6.17.6.4.4 各程序可提取的信息

- Gaussian：保留泛函或波函数方法、基组、完整方法表达式、第二层方法与基组、density-fitting 基组、溶剂、色散、积分格点、SCF/优化收敛、TD/TDA 的态数与 root，以及 `--Link1--` 阶段。`EmpiricalDispersion=GD3BJ`、`Opt=Tight` 等 flag-style 选项按选项名称解释；布尔解析状态不作为方法参数写入正文。
- ORCA：读取结构化 simple keywords 和 `%tddft`、`%cis`、`%cpcm`、`%scf` 等 input block；识别主方法、轨道/辅助基组、RI 近似、色散、溶剂、收敛设置、激发态态数与 root。
- AMESP：读取 method line、坐标区电荷/多重度，以及 `>method`、`>posthf`、`>scf`、`>opt`、`>freq`、`>solvent` 等 section 中的相关参数。
- XTB：读取 GFN 方法、优化或 Hessian/梯度/MD 操作、电荷、未配对电子数、ALPB/GBSA 模型与溶剂。

6.17.6.4.5 输出与可追溯性

`method.md` 以 `DFT Calculations`、`Electronic Structure Calculations` 和 `Excited-State Analysis` 等章节组织可直接编辑的方法学文字。Gaussian 软件名与程序引用集中在软件总述句；同一段中的计算句不重复程序名。总电荷、自旋多重度、激发态数量和 root 不写

入常规方法正文。**method.json** 同时保存结构化 workflow，并在 step attrs 和 raw facts 中保留这些可追溯元数据。每个 workflow step 的 provenance 包括任务名、程序、任务目录、来源类型、来源文件、可信度和原始解析事实。Gaussian 多 job 输入还包含 **source_stage_index** 与 **source_stage_count**，用于定位步骤来自同一输入的哪个阶段。

频率任务若能从结果快照读取虚频数，会区分局部极小值、一级过渡态和高阶鞍点；没有可靠结果时只说明执行了频率计算，不推断驻点类型。

补充说明：

- 自动路线采集覆盖 Gaussian、ORCA、AMESP 与 XTB。其他程序类型保留在任务计划中，并跳过无法验证的方法路线构造。
- 参考库可通过 **btproc method refs lint --refs <file>** 检查完整性。
- 若通过 **finalize** 调用，**BANETASK_BTPROC_PATH** 可覆写可执行文件路径；参考库仍由 **btproc method** 按上述顺序解析，除非在 **.bt** 中显式配置 **finalize.method.refs**。

6.17.6.5 btproc fcclasses report

btproc fcclasses report 可在 task、case 或 project 作用域读取 FCclasses 计划与结果。task 作用域使用 **--task-dir <dir>**；case/project 使用 **--scope case|project --path <root>**。**--out <file>** 写文本报告，**--json <file>** 写 JSON，**--tsv <file>** 写 TSV，**--print** 同时输出文本到标准输出。

6.17.7 var

var 读取、写入、删除、筛选和导出 YAML 标量。目标为 **.bt** 时，只处理第一个任务块之前的 YAML 头部；任务块正文不参与变量操作。

6.17.7.1 用法

```
var write <key> --value <value> [--file|-f <file>]
var write <key> --exec <command> [--file|-f <file>]
var read <key> [--file|-f <file>]
var take <key> [--file|-f <file>]
var list [--file|-f <file>]
var grep <key-pattern> [--value <value-pattern>] [--file|-f <file>]
var csv --region|-r [pattern] [--file|-f <file>] [--output|-o <file>]
var csv --normal|-n [pattern] [--file|-f <file>] [--output|-o <file>]
var help
```

--file / **-f** 省略时使用 **BANEVAR_YAMLFILE**。两者都未提供时返回非零。

6.17.7.2 键路径与值

- 键路径可用冒号或点分隔：**region1:a** 与 **region1.a** 指向同一标量。
- **list**、**grep** 和写入确认信息统一以冒号分隔输出，例如 **define:func=B3LYP**。
- **write --value** 直接写字符串；**write --exec** 执行 shell 命令，把标准输出去掉末尾换行后写入。命令失败时不修改文件。
- 一次写入宜在 **--value** 与 **--exec** 中选择一个，并且最终结果不能是空字符串。两者同时给出时，无论参数顺序如何，**--exec** 的命令输出覆盖 **--value**。
- 已存在的 key 会被覆盖，并在标准错误中给出警告。
- 普通 YAML 文件不存在时，**write** 会创建父目录和初始文件；**.bt** 必须已经存在。

- **read** 只输出值。**take** 先输出值，再从文件删除该 key。不存在的 key 返回非零。
- 工具只处理 scalar 和嵌套 map；列表或其他复合节点不作为扁平 key/value 输出。

6.17.7.3 .bt 文件处理

.bt 头部按 YAML 读取。写入或删除时保留 shebang、头部 YAML 之前的注释以及第一个 **$** 起的任务块正文；YAML 数据区会按规范 YAML 重新序列化。因此，不应依赖 YAML 数据区写入前的空白或键顺序。

6.17.7.4 通配筛选

grep 对扁平化后的完整 key 做全串匹配；**--value** 对完整值做全串匹配。模式中：

- ***** 匹配任意长度片段；
- **[]** 与 ***** 等价；
- 其他字符按字面量匹配，包括 **.**, **?**, **+**, **(**, **)** 等正则特殊字符。

没有匹配项时，标准输出为空，标准错误输出 **没有找到匹配的变量**，命令仍返回 **0**。脚本若把“至少匹配一项”视为成功条件，应检查标准输出，而不能只检查退出码。

```
var grep 'region*:energy' --file vars.yaml
var grep 'ISC_properties[]:T[]:needs_kisc_calculation' --value true --file
↪  soc.yaml
```

6.17.7.5 CSV 导出

- **--region** / **-r** 把第一层 region map 导出为“region 为行、字段为列”的表。
- **--normal** / **-n** 只导出根级 scalar，列为 **Variable,Value**。
- 模式参数可紧跟在模式选项后，使用与 **grep** 相同的 ***** / **[]** 规则。
- 至少选择一种模式。两种模式同时出现时，命令从左到右解析，由后出现的 **--region** 或 **--normal** 决定最终模式及模式参数；脚本宜只给出一种。
- **--output** / **-o** 指定输出；省略时文件名为 **<输入 stem>_<region|normal>_<YYYYMMDD_HHMMSS>.csv**，写到当前工作目录。
- 导出同时在终端打印表格和最终 CSV 路径。若模式没有匹配项，标准错误分别输出 **没有找到匹配的 region** 或 **没有找到匹配的普通变量**，不创建 CSV；命令仍返回 **0**，并仍会打印 **CSV 已保存到: <path>**。自动化脚本应额外检查输出文件是否存在。

6.17.7.6 示例

```
var write define:func --value B3LYP --file task.bt
var write build:revision --exec 'git rev-parse --short HEAD' --file vars.yaml
var read define.func --file task.bt
var take region1:temporary --file vars.yaml
var list --file vars.yaml
var grep 'region[]:energy' --value '-*' --file vars.yaml
var csv --region 'rodft_0[]' --file vars.yaml
var csv --normal 'method*' --file vars.yaml --output methods.csv
```